openMairie Framework Documentation

Version 4.4

openMairie

08 04 2014

Note : Cette création est mise à disposition selon le Contrat Paternité-Partage des Conditions Initiales à l'Identique 2.0 France disponible en ligne http ://creativecommons.org/licenses/by-sa/2.0/fr/ ou par courrier postal à Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

Ce document a pour but de guider les développeurs dans la mise en œuvre d'un projet openMairie.

Avec plus de 30 applications développées pour les collectivités locales accessibles sur le site http ://openmairie.org, nous souhaitons au travers de ce guide , diffuser notre expérience auprès des collectivités et des acteurs économiques du libre qui les accompagnent.

C'est donc une méthode conçue au fur à mesure de nos développements que nous vous proposons de partager et toutes remarques sont les bienvenues, alors n'hésitez pas à nous en faire part à l'adresse mail suivante : contact@openmairie.org

Nous avons conçu openMairie pour fabriquer une maquette le plus en amont possible en s'appuyant sur le modèle de données créé dans la base de données et en intégrant les composants standards du "monde libre".

Cette maquette permet très rapidement d'engager un dialogue participatif avec les utilisateurs, de concentrer le développeur uniquement sur le "métier" et de faire valider par l'utilisateur les évolutions successives.

Si vous débutez, il est préférable de commencer par le chapitre "créer une application" qui permet de prendre en main facilement le générateur et le framework openMairie en vous guidant pas à pas dans le mise en place d'une gestion de courrier.

Le chapitre sur le "framework" complète l'exemple ci dessus en vous décrivant le paramétrage, les classes formulaires et éditions du framework. Il a pour but de vous informer de manière complète sur le fonctionnement du framework

Le chapitre consacré au "générateur" décrit dans le détail le fonctionnement de cet outil et de ses assistants. Cet outil permet de fabriquer la maquette.

La version 4.1.0 permet de construire des applications composites (ou mash up) en intégrant des contenus venant d'application externes. Cela permet de construire rapidement une application à faible coût grâce à la fusion de multiple service internet

Le chapitre consacré a l' "information géographique" décrit dans le détail le fonctionnement SIG interne d'openMairie combinant les API d'internet avec le framework. (La version 4.2.0 améliore l'interface avec l'intégration de web service)

Enfin ce document rassemble toutes les règles de codage du projet openMairie, ainsi que des outils pour aider et guider les développeurs de la communauté.

Les règles indiquées doivent être appliquées pour qu'un projet puisse intégrer la distribution openMairie car l'objectif est de faciliter la lisibilité et la maintenance du code ainsi que la prise en main par les collectivités.

Bonne lecture !

Tutoriel - Créer une application

1.1 Tutoriel - Créer une application

Ce chapitre vous propose de créer une application de gestion de courrier pas à pas.

1.1.1 Créer la base de données

Vous devez au préalable récupérer le framework. Dans le repertoire www de votre serveur apache :

```
svn checkout svn://scm.adullact.net/scmrepos/svn/openmairie/openmairie_exemple/trunk openExemple
```

Il vous est proposé de créer la base de données sous PostgreSQL :
- Créer les tables nécessaires au framework openMairie :

```
cd data/pgsql
sudo su postgres
dropdb openexemple && createdb openexemple && psql openexemple -f install.sql
```

- Créer les tables nécessaires à notre exemple :
  - table courrier

```
courrier        int 8       cle primaire
dateenvoi       date
objetcourrier   text
emetteur        int8        cle secondaire
service         int8        cle secondaire
```

  - table emetteur

```
emetteur        int 8       cle primaire
nom             varchar 20
prenom          varchar 20
```

  - table service

```
service         int 8        cle primaire
libelle         varchar 20
```

  - La requête correspondante en PostgreSQL est la suivante :

```
-- Création des séquences

CREATE SEQUENCE emetteur_seq
START WITH 1
INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
CACHE 1;
```

```
CREATE SEQUENCE service_seq
START WITH 1
INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
CACHE 1;

CREATE SEQUENCE courrier_seq
START WITH 1
INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
CACHE 1;

-- Création des tables

CREATE TABLE emetteur (
    emetteur          int PRIMARY KEY,  -- clé primaire
    nom               varchar(20),
    prenom            varchar(20)
);

CREATE TABLE service (
    service           int PRIMARY KEY,  -- clé primaire
    libelle           varchar(20)
);

CREATE TABLE courrier (
    courrier          int PRIMARY KEY,           -- clé primaire
    dateenvoi         date,
    objetcourrier     text,
    emetteur          int REFERENCES emetteur,   -- clé étrangère
    service           int REFERENCES service     -- clé étrangère
);
```

– Modifier le paramétrage openMairie pour faire un accès à la base créée :

  dyn/database.inc.php
  voir framework/parametrage

– Accéder avec votre navigateur sur openExemple :

  login : **demo**
  mot de passe : **demo**

1.1.2 Créer les formulaires

Nous allons maintenant créer les formulaires à l'aide du générateur.

Pour cela, il faut aller dans le menu **Administration -> Générateur**.

Vous devez avoir 3 nouveaux boutons : courrier, service et emetteur.

Avant de commencer, l'utilisateur apache **www-data** doit avoir les droits d'écriture dans les repertoires */gen* , */sql* et */obj*.

1.1.2.1 Générer les formulaires et édition du courrier

En appuyant sur le bouton de courrier, vous avez les choix de génération :

Choix des fichiers à générer

Selection	Nom Fichier	Generer
		formulaire
☑	courrier.inc.php	../gen/sql/pgsql/courrier.inc.php [Le fichier n'existe pas ou n'est pas accessible]
☑	courrier.inc.php	../sql/pgsql/courrier.inc.php [Le fichier n'existe pas ou n'est pas accessible]
☑	courrier.form.inc.php	../gen/sql/pgsql/courrier.form.inc.php [Le fichier n'existe pas ou n'est pas accessible]
☑	courrier.form.inc.php	../sql/pgsql/courrier.form.inc.php [Le fichier n'existe pas ou n'est pas accessible]
☑	courrier.class.php	../gen/obj/courrier.class.php [Le fichier n'existe pas ou n'est pas accessible]
☑	courrier.class.php	../obj/courrier.class.php [Le fichier n'existe pas ou n'est pas accessible]
		édition
☐	courrier.pdf.inc.php	../sql/pgsql/courrier.pdf.inc.php [Le fichier n'existe pas ou n'est pas accessible]
		reqmo
☐	courrier.reqmo.inc.php	../sql/pgsql/courrier.reqmo.inc.php [Le fichier n'existe pas ou n'est pas accessible]
☐	courrier_emetteur.reqmo.inc.php	../sql/pgsql/courrier_emetteur.reqmo.inc.php [Le fichier n'existe pas ou n'est pas accessible]
☐	courrier_service.reqmo.inc.php	../sql/pgsql/courrier_service.reqmo.inc.php [Le fichier n'existe pas ou n'est pas accessible]
		divers
☐	courrier.import.inc.php	../sql/pgsql/courrier.import.inc.php [Le fichier n'existe pas ou n'est pas accessible]

Générer les fichiers de la table : 'courrier' **Retour**

Au préalable, le générateur a fait une analyse de la base de données :

analyse

Analyse de la base de données pgsql → openexemple.openexemple

Elements	Infos
Tables de la base de données	[om_lettretype] [om_profil] [om_collectivite] [om_sousetat] [om_logo] [om_utilisateur] [om_parametre] [om_dashboard] [om_sig_map_comp] [emetteur] [om_droit] [service] [om_sig_map_wms] [om_requete] [om_sig_wms] [om_widget] [om_sig_map] [om_etat]
Table : **courrier**	[clé N - clé automatique] **[courrier]** [longueur enregistrement : 65]
Champs	[courrier 11 int] [dateenvoi 12 date] [objetcourrier -5 blob] [emetteur 11 int] [service 11 int] [registre 20 string]
Sous-formulaire	
Clé secondaire	[emetteur] [service]

Le générateur a donc détecté 2 clés secondaires et aucun sous formulaire.

C'est pour cela qu'il propose 3 "reqmo" : 1 "reqmo" global et 2 "reqmos" suivant la clé secondaire.

Par défaut, seules les options du formulaire sont cochées.

Si vous le refaites plus tard, seules celles frabriquées par le générateur seront cochées.

Cochez les toutes :

- ☑ courrier.inc.php
- ☑ courrier.inc.php
- ☑ courrier.form.inc.php
- ☑ courrier.form.inc.php
- ☑ courrier.class.php
- ☑ courrier.class.php
- ☑ courrier.pdf.inc.php
- ☑ courrier.reqmo.inc.php
- ☑ courrier_emetteur.reqmo.inc.php
- ☑ courrier_service.reqmo.inc.php
- ☑ courrier.import.inc.php

En cliquant sur valider, vous avez le message :

Le paramétrage utilisé est le paramétrage standard.

Vous pouvez le modifier : *voir générateur/paramétrage*.

L'affichage par colone est "ok", ce qui veut dire que la taille des colones dans le fichier pdf sera complet (attention le script ne prend pas le champ blob).

1.1.2.2 Générer les formulaires et édition de l'emetteur

Nous allons procéder de la même manière avec le bouton emetteur.

L'analyse de la base de données est la suivante :

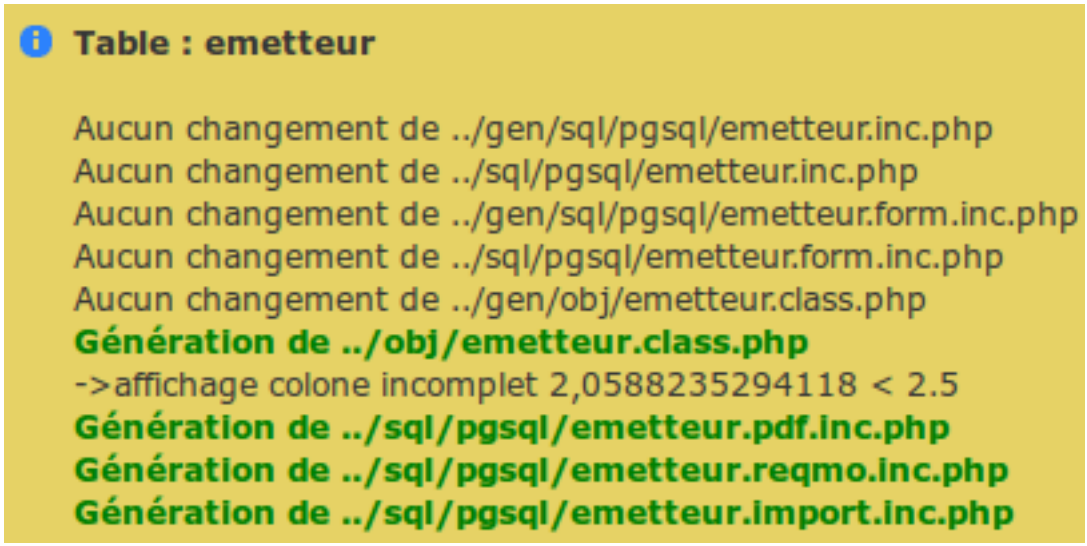
analyse

Analyse de la base de données pgsql openexemple.openexemple

Elements	Infos
Tables de la base de données	[om_lettretype] [om_profil] [om_collectivite] [om_sousetat] [om_logo] [om_utilisateur] [om_parametre] [om_dashboard] [om_sig_map_comp] [courrier] [om_droit] [service] [om_sig_map_wms] [om_requete] [om_sig_wms] [om_widget] [om_sig_map] [om_etat]
Table : **emetteur**	[clé N - clé automatique] **[emetteur]** [longueur enregistrement : 136]
Champs	[emetteur 11 int] [nom 20 string] [prenom 20 string] [adresse 40 string] [cp 5 string] [ville 40 string]
Sous-formulaire	[courrier]
Clé secondaire	

Le générateur repère un sous formulaire courrier. Effectivement, il y a une relation de un à plusieurs entre emetteur et courrier : un emetteur peut avoir 0 à plusieurs courriers.

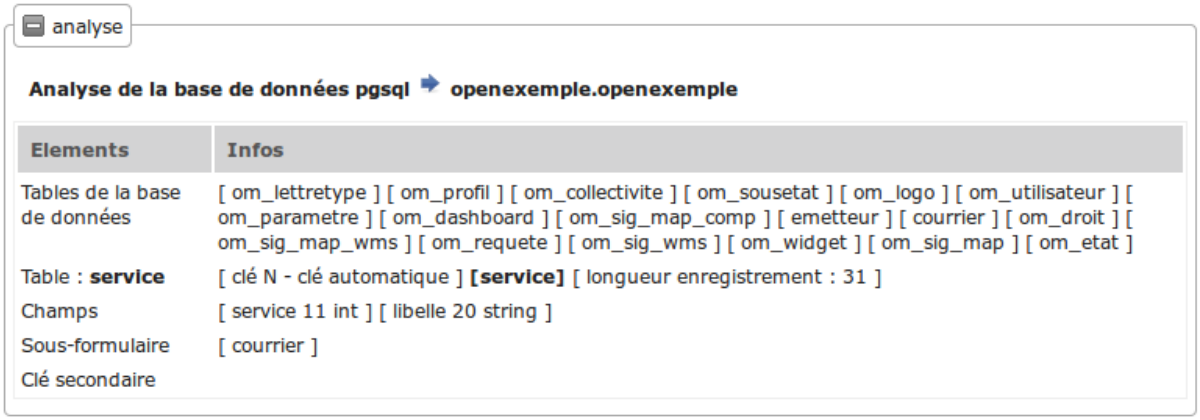
En cliquant sur toutes les options puis en validant, vous avez le message suivant :



1.1.2.3 Générer les formulaires et édition de service

Nous allons procéder de la même manière avec le bouton service

L'analyse de la base de données est la suivante :


analyse
Analyse de la base de données pgsql openexemple.openexemple
Elements
Infos
Tables de la base de données
[om_lettretype] [om_profil] [om_collectivite] [om_sousetat] [om_logo] [om_utilisateur] [om_parametre] [om_dashboard] [om_sig_map_comp] [emetteur] [courrier] [om_droit] [om_sig_map_wms] [om_requete] [om_sig_wms] [om_widget] [om_sig_map] [om_etat]
Table : service
[clé N - clé automatique] [service] [longueur enregistrement : 31]
Champs
[service 11 int] [libelle 20 string]
Sous-formulaire
[courrier]
Clé secondaire

Le générateur repère un sous formulaire courrier. Effectivement, il y a une relation de un à plusieurs entre service et courrier : un service peut avoir 0 à plusieurs courriers.

En cliquant sur toutes les options, vous avez le message suivant :

Table : service
Génération de ../gen/sql/pgsql/service.inc.php
Aucun changement de ../sql/pgsql/service.inc.php
Aucun changement de ../gen/sql/pgsql/service.form.inc.php
Aucun changement de ../sql/pgsql/service.form.inc.php
Aucun changement de ../gen/obj/service.class.php
Aucun changement de ../obj/service.class.php
->affichage colone ok 9,0322580645161 >= 2.5
Aucun changement de ../sql/pgsql/service.pdf.inc.php
Aucun changement de ../sql/pgsql/service.reqmo.inc.php
Aucun changement de ../sql/pgsql/service.import.inc.php

1.1.2.4 Intégrer les formulaires dans le menu

Pour accéder à nos formulaires, nous allons les intégrer dans le menu (*voir framework/parametrage/menu gauche*).

Nous allons appeller le formulaire depuis le menu :
– option **Application** -> tab.php ?obj=courrier
– option **Paramétrage** -> tab.php ?obj=emetteur
– option **Paramétrage** -> tab.php ?obj=service
Il faut ouvrir avec un éditeur le fichier *dyn/menu.inc.php* et insérer le code suivant :

```
// {{{ Rubrique APPLICATION

    $links[] = array(
        "href" => "../scr/tab.php?obj=courrier",
        "class" => "courrier",
        "title" => _("courrier"),
        "right" => "courrier"
    );

// {{{ Rubrique PARAMETRAGE

    $links[] = array(
        "href" => "../scr/tab.php?obj=emetteur",
        "class" => "emetteur",
        "title" => _("emetteur"),
        "right" => "emetteur"
    );

    $links[] = array(
        "href" => "../scr/tab.php?obj=service",
        "class" => "service",
        "title" => _("service"),
        "right" => "service"
    );
```

Il faut également bien placer le code, c'est à dire dans la bonne rubrique (précisée en commentaire) après

```
$links = array();
```

et avant

```
$rubrik['links'] = $links;
```

Enfin pour y accéder il faut soit donner les droits via le menu framework, soit (et c'est en l'occurrence le cas) dans le fichier config.inc.php (option utilisée que pour le développement) décommenter la ligne

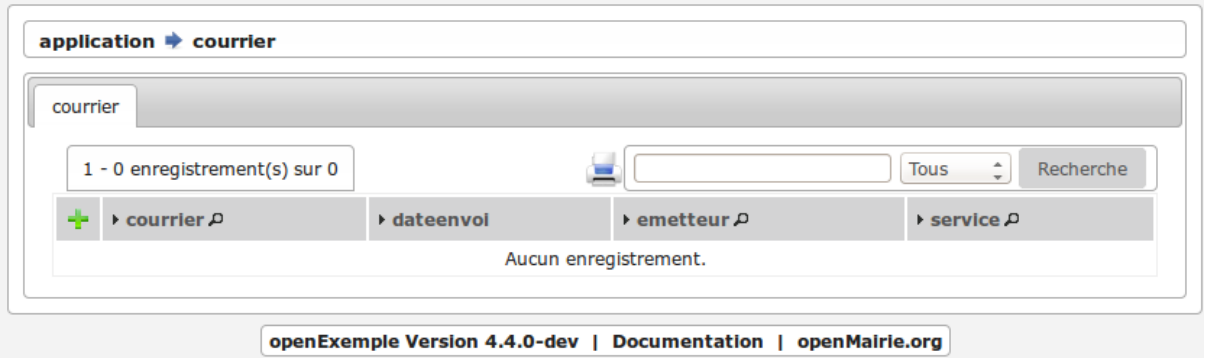
```
$config['permission_if_right_does_not_exist'] = true;
```

Vous pouvez maintenant accéder à vos formulaires par le menu.

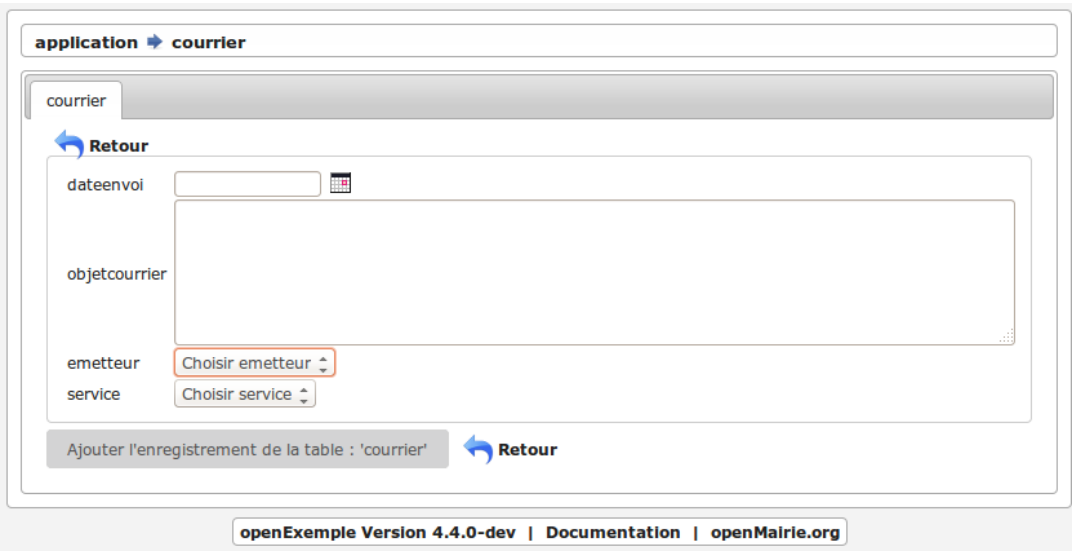
1.1.2.5 Menu

Application -> Courrier

Cette opération affiche la table courrier :

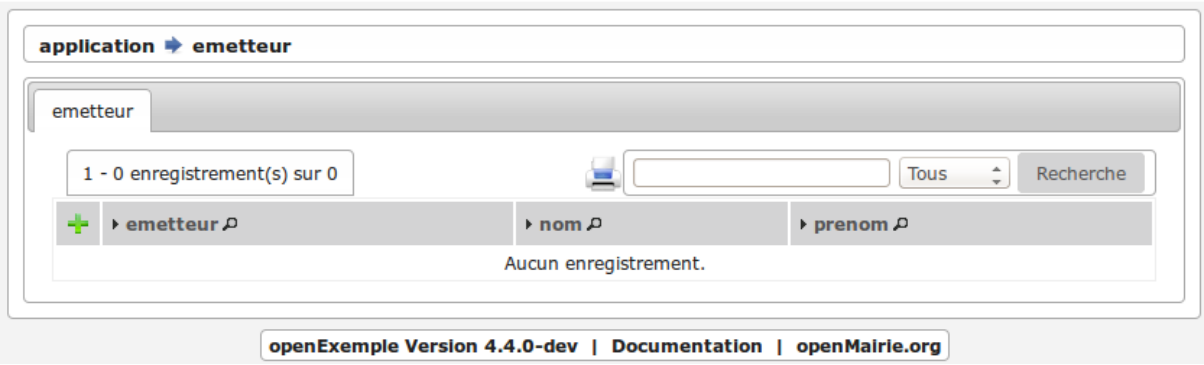


On accéde en appuyant sur **+** au formulaire d'insertion où les champs sont :
– la date du courrier avec calendrier,
– l'objet du courrier dans un champ textarea,
– deux contrôles "select" pour le service et l emetteur.



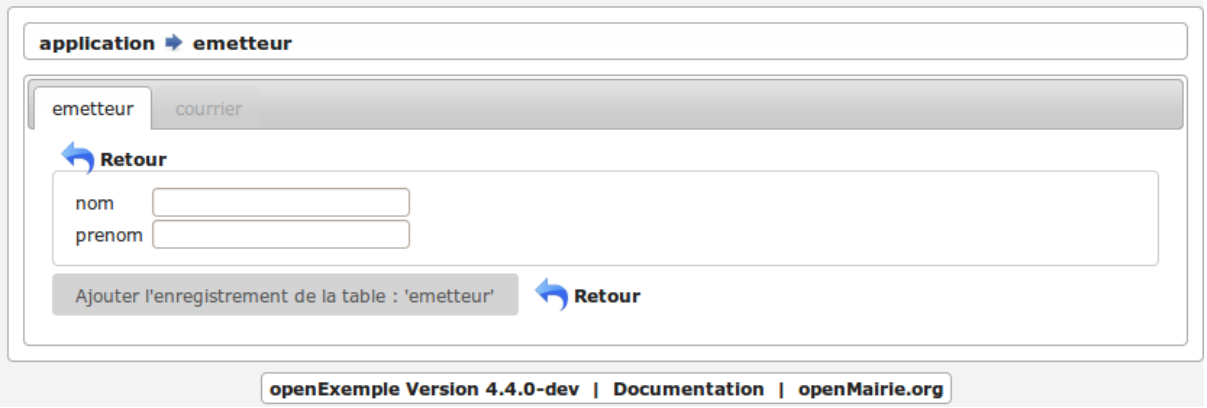
Paramétrage -> Emetteur

Cette operation affiche la table emetteur :



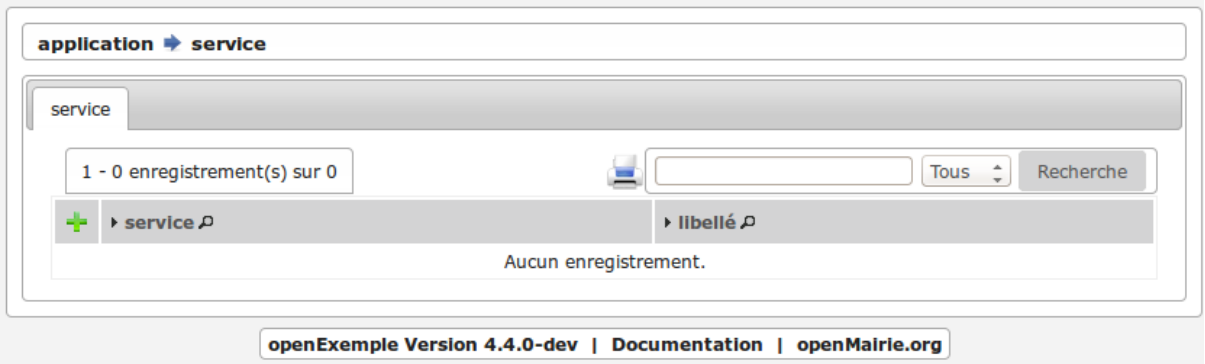
En appuyant sur **+**, on accède à la saisie.

L'onglet courrier est inactif tant que l'emetteur n'est pas saisi et validé.



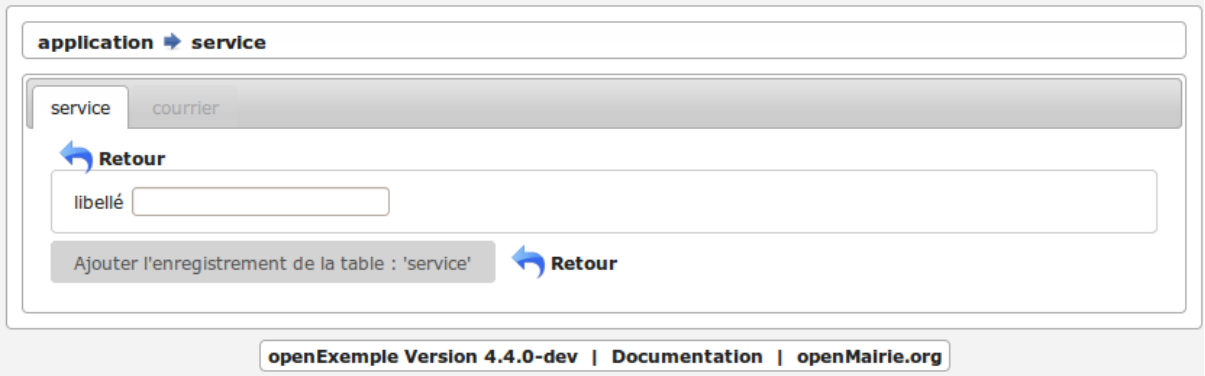
Paramétrage -> Service

Cette opération affiche la table service :



En appuyant sur **+**, on accède à la saisie.

L'onglet courrier est inactif tant que le service n'est pas saisi.



Vous pouvez accéder aux éditions et requêtes mémorisées :

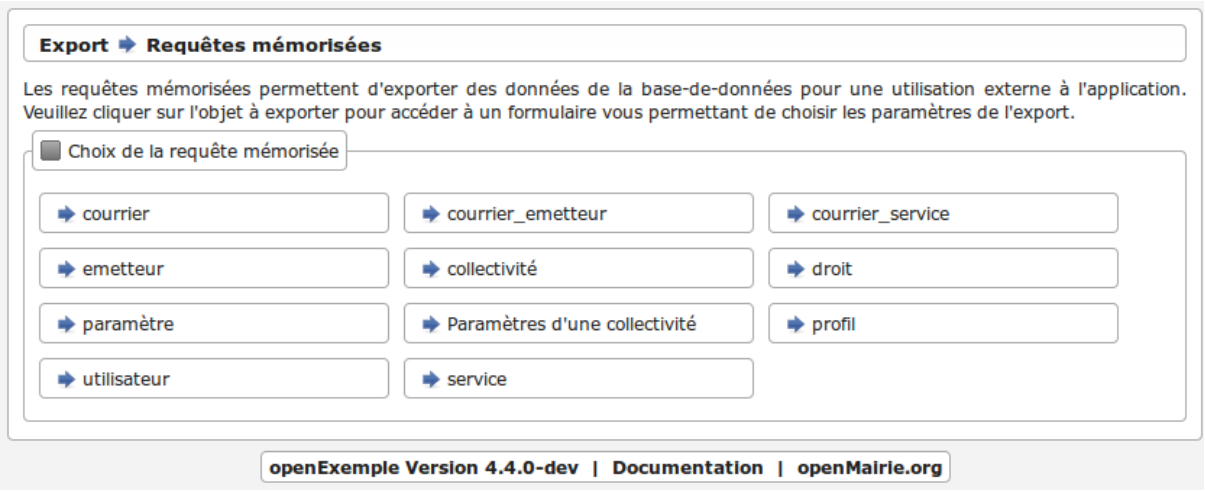
Export -> Edition

Cet option affiche l'ensemble des éditions pdf :

pour en savoir plus voir *framework/edition*

Export -> Requêtes Mémorisées

Cette option affiche les requêtes mémorisées :



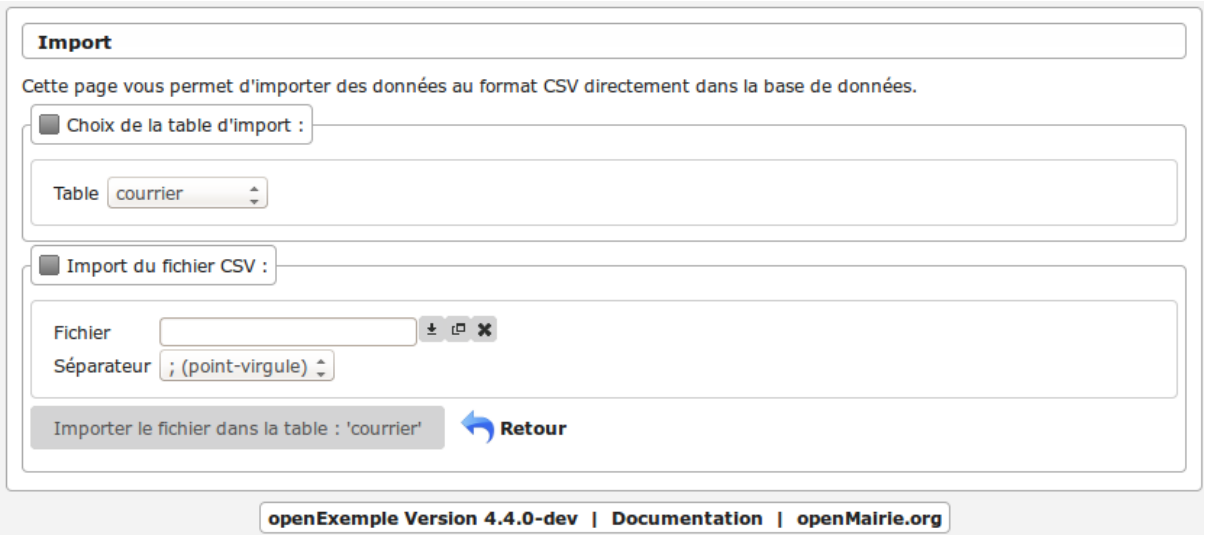
pour en savoir plus voir *framework/reqmo*

Vous pouvez accéder aux éditions en appuyant dans le formulaire d'affichage sur l'imprimante.

Vous pouvez accéder au fichiers d'import :

Administration -> Import

Cette option affiche les scripts d'imports :


Import
Cette page vous permet d'importer des données au format CSV directement dans la base de données.
Choix de la table d'import :
Table courrier
Import du fichier CSV :
Fichier
Séparateur ; (point-virgule)
Importer le fichier dans la table : 'courrier'
Retour
openExemple Version 4.4.0-dev | Documentation | openMairie.org

Pour en savoir plus *voir framework/import*.

1.1.3 Personnaliser son application

Nous allons maintenant personnaliser notre application.

Pour ce faire, nous allons saisir un jeu de données.

Vous pouvez le faire avec les formulaires, l'incrémentation des séquences étant faite par le framework. Tout comme la création des tables stockant les séquences (méthode *setId* des objets metier).

Sinon exécutez la requête PostgreSQL suivante :

```
-- création des tables de séquence déjà faite

-- insertion de deux émetteurs avec récupération et incrémentation de la table de séquences

INSERT INTO emetteur (emetteur, nom, prenom) VALUES
(nextval('emetteur_seq'), 'dupont', 'pierre'),
(nextval('emetteur_seq'), 'durant', 'jacques');

-- insertion de deux services avec récupération et incrémentation de la table de séquences

INSERT INTO service (service, libelle) VALUES
(nextval('service_seq'), 'informatique'),
(nextval('service_seq'), 'telephonie');

-- insertion de deux courriers avec récupération et incrémentation de la table de séquences

INSERT INTO courrier (courrier, dateenvoi, objetcourrier, emetteur, service) VALUES
(nextval('courrier_seq'), '2010-12-01', 'Proposition de fourniture de service', 1, 1),
(nextval('courrier_seq'), '2010-12-02', 'Envoi de devis pour formation openMairie', 2, 1);
```

1.1.3.1 Faire un affichage courrier plus convivial

L'affichage des courriers se fait avec des libellés générés automatiquement.

Ainsi dans le fichier *gen/sql/pgsql/courrier.inc.php* (qui est inclus dans le fichier *sql/pgsql/courrier.inc.php* que vous pourrez modifier) vous avez la variable **$champAffiche**.

Vu que ce fichier a été créé par le générateur (et est en lecture seule) et vu que nous souhaitons modifier la variable (pour par exemple avoir le nom et le prénom de l'emetteur au lieu de simplement son nom) il nous faut ouvrir le fichier *sql/pgsql/courrier.inc.php* où nous allons (après l'*include* !) réaffecter à la variable **$champAffiche** la valeur suivante :

```
$champAffiche = array(
    'courrier.courrier as "'._("courrier").'"',
    'to_char(courrier.dateenvoi ,\'DD/MM/YYYY\') as "'._("dateenvoi").'"',
    'concat(emetteur.nom,\' \',emetteur.prenom) as "'._("emetteur").'"',
    'service.libelle as "'._("service").'"',
);
```

Il est possible que l'opération vous soit refusée (seul **www-data** ayant les droits d'écriture).

Si tel est le cas échéant il faudra se rajouter les permissions.

Le résultat est le suivant :

```
Courrier Dateenvoi  Emetteur            Service
    1       01/12/2010      dupont pierre   informatique
    2       02/12/2010      durant jacques  informatique
```

De la même manière, toujours dans le même fichier, vous pouvez changer les options de la zone de recherche en réaffectant la variable **$champRecherche**. Actuellement on peut en plus de *Tous* faire une recherche sur *courrier*, *emetteur* et *service*.

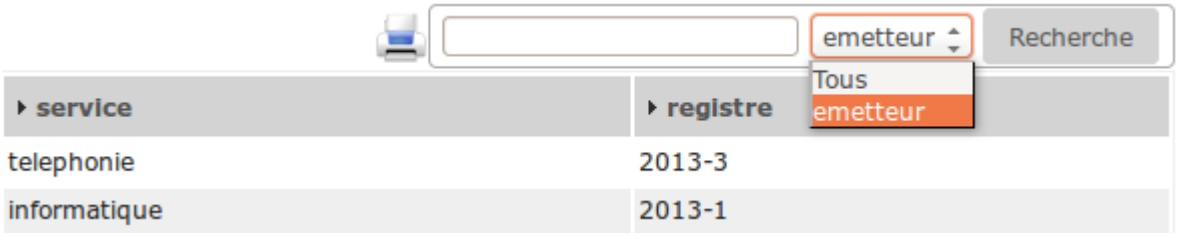
C'est parce qu'à l'orgine, dans le fichier généré, **$champRecherche** avait été affectée comme telle :

```
$champRecherche = array(
'courrier.courrier as "'._("courrier").'"',
'emetteur.nom as "'._("emetteur").'"',
'service.libelle as "'._("service").'"',
);
```

Supprimez un ou plusieurs élément(s) du tableau et il disparaîtra de la zone de recherche. Par exemple...

```
$champRecherche = array(
'emetteur.nom as "'._("emetteur").'"',
);
```

...donnera :



Nous souhaitons maintenant avoir les derniers courriers au début de la page affichée. Nous n'avons pas besoin d'aller réécrire la requête, il existe une variable texte comprenant l'instruction de tri. Réaffectez cette variable **$tri** dans votre *courrier.inc.php* de la manière suivante :

```
$tri= " order by dateenvoi desc";
```

Le résultat est le suivant :

Pour en savoir plus sur ces variables *voir framework/affichage*.

1.1.3.2 Rendre obligatoire des champs

Nous avons affiché le courrier avec une jointure de type *LEFT JOIN* ce qui ne rend pas obligatoire la saisie de l'emetteur et du service (auquel le courrier est affecté).

Nous devons surcharger la méthode **verifier**().

Dans *obj/courrier.class.php* la méthode à insérer après le constructeur est celle-ci :

```
function verifier($val,&$db,$DEBUG) {
    parent::verifier($val,$db,$DEBUG);

    // Les champs service et emetteur sont obligatoires
    if ($this->valF['service']==""){
        $this->correct=False;
        $this->addToMessage(_('service')." "._('obligatoire')." !");

    //
    }
    if ($this->valF['emetteur']==""){
        $this->correct=False;
        $this->addToMessage(_('emetteur')." "._('obligatoire')." !");
    }
}
```

Par défaut le premier champ (ici *dateenvoi*) est obligatoire, cette option est modifiable dans le générateur.

La commande *parent : :verifier($val,$db,$DEBUG) ;* permet de ne pas neutraliser la fonction surchargée (ici dans gen/obj/courrier.class.php)

Pour plus d'informations *voir framework/methode*.

1.1.3.3 Valoriser un champ par défaut

Pour simplifier la saisie, nous souhaitons mettre la date du jour dans le champ *dateenvoi* lors d'un ajout de courrier.

Nous allons surcharger la methode setVal() dans *obj/courrier.class.php* de la manière suivante :

```
function setVal(&$form, $maj, $validation, &$db, $DEBUG=null){
    parent::setVal($form, $maj, $validation, $db, $DEBUG=null);

    if ($validation==0) {
        if ($maj == 0){
            $form->setVal("dateenvoi", date('Y-m-d'));
        }
    }
}
```

Le champ *dateenvoi* contiendra la date système *(date('Y_m-d'))* si la validation est égale à 0 (ce qui signifie que le formulaire n'a pas été validé) et si **$maj** est égal à 0 (ce qui signifie qu'il s'agit d'un ajout).

Les autres valeurs que peut prendre **$maj** sont :
– 1 : modifier
– 2 : supprimer
– 3 : consulter

1.1.3.4 Mettre en majuscule un champ

Nous souhaitons maintenant mettre en majuscule le champ *nom* de la table emetteur.

Nous allons surcharger la méthode setOnchange() dans *obj/emetteur.class.php* de la manière suivante :

```
function setOnchange(&$form,$maj){
    parent::setOnchange($form,$maj);

    $form->setOnchange("nom","this.value=this.value.toUpperCase()");
}
```

A la saisie ou à la modification du nom, le champ se mettra en majuscule.

1.1.3.5 Principe à retenir

Voila quelques exemples des possibilités de modification dans les fichiers sql (répertoire *sql/...*) et dans les méthodes de l'objet (repertoire *obj/...*).

En aucun cas il ne faut modifier les fichiers dans *gen/* qui est l'espace de travail propre au générateur.

Nous allons dans le prochain chapitre modifier la base et regénérer les écrans sans mettre en danger votre personnalisation.

1.1.4 Modifier la base et regénérer

Le framework openMairie permet de modifier la base et prendre en compte ces modifications en regénérant les scripts **sans** mettre en péril la personnalisation que vous avez effectuée.

Nous vous proposons de rajouter un champ *registre* dans la table *courrier* et de rajouter l'adresse dans la table *emetteur*.

1.1.4.1 Rajouter un champ registre dans courrier

Il est proposé de rajouter un champ *registre* dans le courrier dont le but est de stocker le numéro de registre du courrier sous la forme *annee_numero_d_ordre*.

Nous allons d'abord créer un champ *registre* dans la table *courrier* de la manière suivante :

```
ALTER TABLE courrier ADD registre VARCHAR( 20 ) ;
```

Vous devez regénérer votre application courrier dans l'option du menu **Administration -> Générateur -> Courrier** et laisser cochées les options par défaut :

gen/obj/courrier.class.php
gen/sql/pgsql/courrier.inc.php
gen/sql/pgsql/courrier.form.inc.php

Validez l'opération.

Vous pouvez remarquer si vous allez sur le formulaire d'ajout qu'il y a un nouveau champ *registre*. Votre personnalisation n'est pas affectée.

Nous voulons que le numéro de registre se mette en ajout de manière automatique une fois le formulaire validé.

Il faut donc surcharger les méthodes suivantes dans *obj/courrier.class.php* :

```
// pour que registre ne soit pas modifiable

function setType(&$form,$maj) {
    parent::setType($form,$maj);
    $form->setType('registre', 'hiddenstatic');
}

// pour la mise à jour de la séquence avant l'ajout de l enregistrement

function triggerajouter($id,&$db,$val,$DEBUG) {
    //  prochain numero de registre
    //  fonction DB pear
    $temp= $db->nextId("registre");
    // fabrication du numero annee_no_d_ordre
    $temp= date('Y')."-".$temp;
    $this->valF['registre'] = $temp;
}
```

Si vous souhaitez que registre apparaisse dans l'affichage de la table, vous devez aussi modifier la variable *champAffiche* de *sql/pgsql/courrier.inc* de la manière suivante :

```
$champAffiche = array(
    'courrier.courrier as "'._("courrier").'"',
    'to_char(courrier.dateenvoi ,\'DD/MM/YYYY\') as "'._("dateenvoi").'"',
    'concat(emetteur.nom,\' \',emetteur.prenom) as "'._("emetteur").'"',
    'service.libelle as "'._("service").'"',
    'registre'
);
```

Votre affichage de la table courrier est modifié.

1.1.4.2 Rajouter l'adresse dans emetteur

Il est proposé de rajouter l'adresse de l'emetteur à savoir : le libellé, le code postal et la ville.

La requête est la suivante :

```
ALTER TABLE emetteur ADD adresse VARCHAR( 40 ) ,
ADD cp VARCHAR( 5 ) ,
ADD ville VARCHAR( 40 ) ;
```

Vous devez regénérer votre application courrier en allant dans l'option du menu : administration -> generateur -> emetteur et laisser cochées les options par défaut :

gen/obj/emetteur.class.php
gen/sql/pgsql/emetteur.inc.php
gen/sql/pgsql/emetteur.form.inc.php

Validez l'opération.

N'ayant pas modifié *sql/pgsql/emetteur.inc*, le framework fonctionne avec le code généré.

1.1.4.3 Améliorer la présentation du formulaire emetteur

Nous pouvons continuer à améliorer les présentations de nos formulaires en utilisant les méthodes setGroupe() et setRegroupe() dans le script *obj/emetteur.class.php*.

Il vous est proposé d'insérer dans votre script obj/emetteur.class.php le code suivant :

```
function setLayout(&$form, $maj) {

    $form->setFieldset('nom','D',_('nom'),"collapsible");
    $form->setFieldset('prenom','F');

    $form->setFieldset('adresse','D',_('adresse'),"startClosed");
    $form->setFieldset('ville','F');
}
```

Le fieldset nom est affiché par défaut, pas celui de l'adresse :

Vos formulaires sont maintenant au point.

Le paragraphe suivant vous indique les surcharges d'openCimetiere que vous pouvez intégrer dans votre exemple, maintenant que vous avez la méthode.

1.1.4.4 Les surcharges d'openCimetiere

Vous pouvez utiliser openCimetiere (actuellement version 3.0.0-a5-dev) qui est téléchargeable au lien suivant :

svn ://scm.adullact.net/scmrepos/svn/opencimetiere/trunk

La base de données d'openCimetiere est plus complexe.

Si les surcharges qui ont été faites dans notre exemple sont celles d'openCourrier, il y a d'autre surcharges dans le script *courrier.class.php* d'openCimetiere :

Les méthodes setLib, setGroupe et setRegroupe permettent **une présentation en fieldset** du courrier (utilisation des champs vide 1 à 5 voir *sql/pgsql/courrier.form.inc*).

Il y a d'autres objets métier qui ont des surcharges intéressantes, par exemple l'objet *obj/dossier.class.php* où vous avez un upload pour télécharger des fichiers.

Vous pouvez regarder également l'application openCourrier mais attention à la base de données qui est en MySQL :

– openCourrier fonctionne avec des restrictions d'accès par service et les méthodes de login ont été modifiées dans *obj/utils.class.php* ainsi qu'*utilisateur.class.php* qui a dans openCourrier un champ service.
– l'objet *obj/tachenonsolde.class.php* est un exemple de surcharge de *tache.class.php* qui affiche que les tâches non soldées

– vous pouvez aussi regarder deux scripts de traitement :
  – trt/num_registre.php qui remet à 0 le numéro de registre
  – trt/archivage.php qui tranfere en archive les courriers avant une date

Vous avez également des détails sur les traitements dans le chapitre *framework/util* notament sur la mise à jour du registre.

1.1.5 Créer ses états

Il vous est proposé de créer un état des courriers par service.

Il sera utilisé dans ce chapitre l'assistant état et sous-état du générateur.

Quittez le projet openCimetiere et revenez à openExemple.

1.1.5.1 Créer l'état service

Nous allons utiliser l'assistant état du générateur dans le menu :

Administration -> Générateur -> Assistants

Choisir *Création d'état* puis choisir dans le select l'option service.

Ensuite avec la touche *CTRL* sélectionner les champs *service.service* et *service.libellé*.

Cliquer ensuite sur *Import service dans la base*.

Administration ➧ Générateur ➧ état
cet assistant vous permet de créer des états directement à partir de vos tables
Choix table :
fichier service
choix des champs
Utilisez ctrl key pour choix multiple
service.service
service.libelle
Import service dans la base
openExemple Version 4.4.0-dev | Documentation | openMairie.org

Un message apparaît : *service enregistré*.

Vous avez créé un enregistrement qui a pour identifiant *service* dans la table *om_etat*.

SELECT * FROM "om_etat" LIMIT 50 Modifier

modifier	om_etat	om_collectivite	id	libelle	actif
	1	1	om_collectivite	om_collectivite gen le 12/11/2010	t
	2	1	service	service gen le 22/11/2013	f

Vous devez rendre d'abord votre état service *actif* pour pouvoir y accéder.

Il faut maintenant permettre l'accès dans l'affichage du service :
– Ouvrir le fichier *sql/pgsql/service.inc.php*
– Ajouter le script suivant :

```
$href[3] = array(
    "lien" => "../pdf/pdfetat.php?obj=".$obj."&amp;idx=",
    "id" => "",
    "lib" => "<img src=\"../om-theme/img/pdf-16x16.png\" alt=\""
             ._("Edition PDF")."\" title=\""._("Edition PDF")."\" />",
);
```

Nous rajoutons la ligne 3 dans le tableau href. Vous avez un état lié à l'affichage du service.

Il y a des exemples d'utilisation de href dans *om_collectivité*, *om_etat*, *om_utilisateur*,...

1.1.5.2 Créer le sous-état courrier

Nous allons utiliser l'assistant sous-état du générateur dans le menu :

Administration -> Générateur -> Assistants -> Création sous-état

Nous choisissons la table courrier et nous surlignons les champs *courrier.dateenvoi*, *courrier.objetcourrier*, *courrier.emetteur* et *courrier.registre*.

Nous choisissons *courrier.service* comme clé secondaire pour faire le lien avec service.

Administration ➧ Générateur ➧ Sous-état
cet assistant vous permet de créer des sous-états directement à partir de vos tables
Choix table :
fichier courrier
choix des champs
Utilisez ctrl key pour choix multiple
courrier.courrier
courrier.dateenvoi
courrier.objetcourrier
courrier.emetteur
choisir la clé de sélection courrier.service
Import courrier dans la base
openExemple Version 4.4.0-dev | Documentation | openMairie.org

En cliquant sur *Import courrier dans la base* vous créez un enregistrement ayant pour identifiant *courrier.service* dans la table *om_sousetat* :

SELECT * FROM "om_sousetat" LIMIT 50 Modifier

modifier	om_sousetat	om_collectivite	id	libelle	actif
	1	1	om_parametre.om_collectivite	gen le 12/11/2010	t
	2	1	courrier.service	gen le 22/11/2013	f

1.1.5.3 Associer le sous-état *courrier* à l'état *service*

Vous devez rendre d'abord votre sous-état *courrier.service* actif pour pouvoir l'associer.

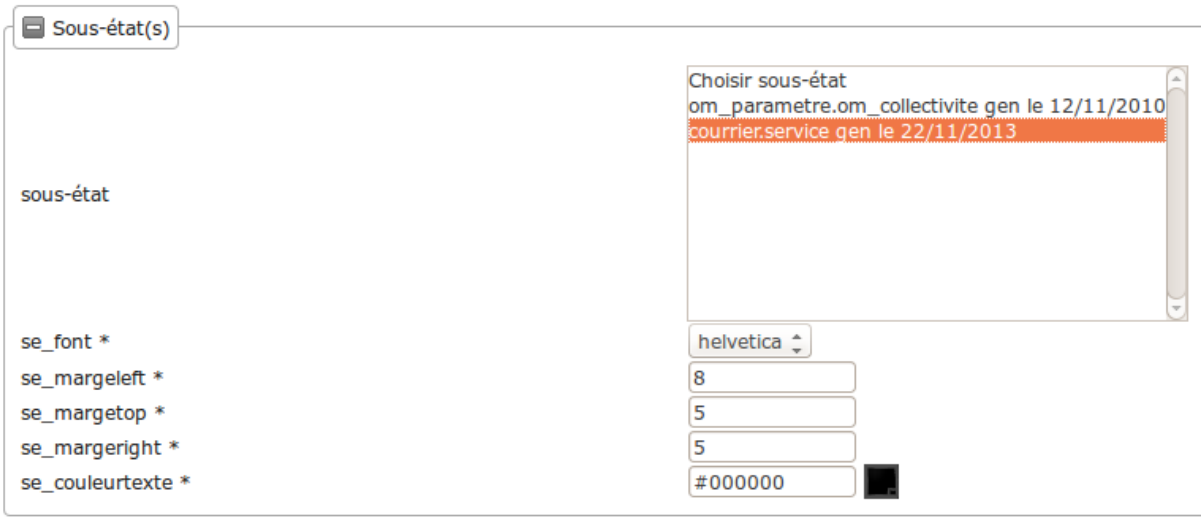
Allez dans l'option **Sous Etat** du menu **Paramétrage**.

Recherchez le sous-état *courrier.service* et modifiez le en cochant actif (premier fieldset : *collectivité*).

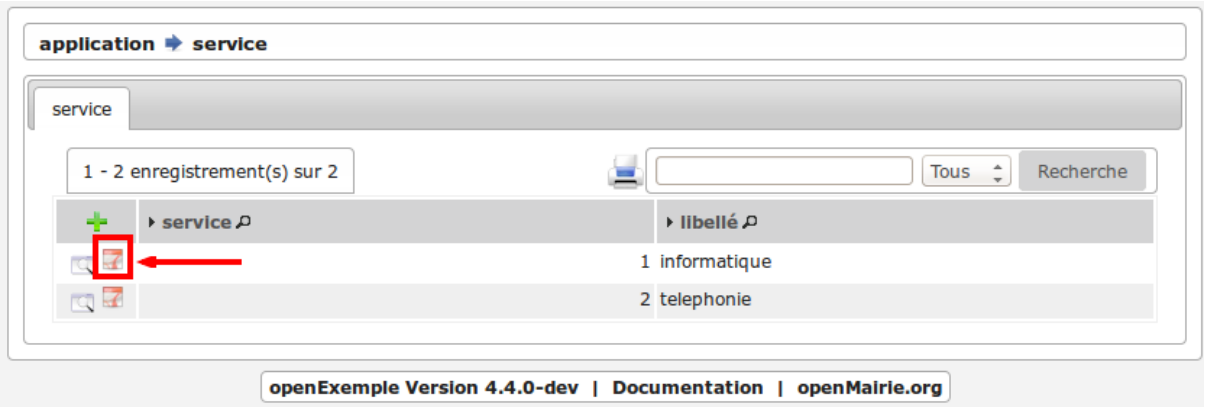
Il vous faut maintenant associer le sous-état *courrier.service* à l'état *service*.

Allez dans l'option **Etat** du menu **Paramétrage**.

Modifiez l'état *service* et dans le fieldset à déplier *Sous-état(s)* après l'avoir coché actif sélectionnez le sous-état *courrier.service*.



Vous avez désormais un état des courriers par service (**Paramétrage -> Service -> Edition PDF**) :



OpenMairie

le 22/11/2013

1
informatique

liste courrier

DATEENVOI	OBJETCOURRIER	EMETTEUR	REGISTRE
2013-11-21	Proposition de fournitures de service.	1	2013-1
2013-11-21	Envoi de devis pour formation openMairie.	1	2013-2

1.1.5.4 Mettre le nom et le prénom de l'emetteur dans le sous-état

Nous souhaitons mettre le nom et le prénom de l'emetteur à la place de la clé secondaire.

Vous devez modifier la requête sql du sous-état *courrier.service* dans la table *om_sousetat* de la manière suivante :

```
select courrier.dateenvoi as dateenvoi,
courrier.objetcourrier as objetcourrier,
concat(emetteur.nom,' ',emetteur.prenom) as emetteur,
courrier.registre as registre
from &DB_PREFIXEcourrier LEFT JOIN &DB_PREFIXEemetteur on emetteur.emetteur = courrier.emetteur
where courrier.service='&idx'
```

Votre nouvel état a la forme suivante :

le 22/11/2013

1
informatique

liste courrier

DATEENVOI	OBJETCOURRIER	EMETTEUR	REGISTRE
2013-11-21	Envoi de devis pour formation openMairie.	dupont pierre	2013-2
2013-11-21	Proposition de fournitures de service.	dupont pierre	2013-1

Vous avez de nombreux exemples d'utilisation d'état et de sous-état dans les applications openMairie.

Une utilisation originale a été faite pour le Cerfa du recensement dans openRecensement où à la place du logo il a été mis une image du Cerfa.

On ne peut cependant pas faire tous les états et il est fort possible que vous ayez des états spécifiques. Vous avez des exemples d'utilisation spécifique des méthodes de fpdf dans openElec : carte électorale, liste électorale,...

Vous pouvez compléter votre information avec le chapitre *framework/edition* et regarder les possibilités de paramétrage du générateur *generateur/parametrage* pour la réalisation d'état customisé.

Vous avez maintenant terminé l'exemple d'utilisation du Framework, le chapitre suivant a pour but de vous informer de manière complète sur ce dernier.

CHAPITRE 2

Le framework

2.1 Le framework

openMairie_exemple est le framework de base dans lequel vous pouvez développer votre propre application.

openMairie_exemple est téléchargeable sur le site de l'adullact.

Il est proposé ici de décrire le fonctionnement du framework.

"En programmation orientée objet un framework est typiquement composé de classes mères qui seront dérivées et étendues par héritage en fonction des besoins spécifiques à chaque logiciel qui utilise le framework".

http ://fr.wikipedia.org/wiki/Framework

Dans un environnement LAMP/WAMP, le framework openMairie intègre des composants au travers de classes qui permettent de créer des formulaires et des états. Ces classes sont surchargées par les objets métier à créer.

openMairie intègre de nombreux composants : DBPEAR et FPDF dans toutes les applications, mais aussi ARTISHOW (pour les graphes), JQUERY pour l'ergonomie, OPENLAYERS pour l interface SIG ...

DBPEAR est un abstracteur de base de données qui permet d'utiliser diverses bases de données notamment MYSQL ou POSTGRESQL.

FPDF est le composant qui permet de gérer le PDF.

Le développement consiste à créer des objets métier qui surchargent la classe abstraite om_dbformdyn.class.php (composant openMairie). De base, les données de la base de données sont récupérées pour le formulaire (longueur, max, nom).
– om_dbformdyn.class.php ; assure la liaison entre le formulaire et la base de données
– om_formulaire.class.php : rassemble toutes les méthodes permettant de construire des formulaires
Ce chapitre propose de vous décrire les outils de base du framework de la manière suivante :
– le paramétrage général du framework en /dyn
– les méthodes pour construire des formulaires avec le framework
– les outils d'édition du framework
– l'outil de requête paramétrable du framework
– la gestion des accès du framework ainsi que la gestion multi-collectivite
– l'ergonomie intégrant jquery
– la gestion de traitement et la construction de programme spécifiques avec les utilitaires
– l'import des données CSV du framework

2.1.1 Arborescence

Cette rubrique vise à décrire brièvement l'arborescence du framework pour comprendre l'objectif de chaque répertoire. Elle est divisée en deux parties : les répertoires spécifiques à l'applicatif qui sont modifiés lors du développement de l'applicatif et les répertoires du framework qui sont récupérés tel quel dans le framework.

`[F] .htaccess` dans les descriptions de répertoires ci-dessous représente des fichiers .htaccess empêchant l'accès dans les répertoires en question qui ne doivent pas être accessibles depuis l'interface par l'utilisateur.

2.1.1.1 Les répertoires spécifiques à l'applicatif

Ces répertoires sont ceux qui font l'applicatif. Par exemple, sur un applicatif comme openCimetière ce sont les répertoires qui vont se trouver dans le gestionnaire de fichiers que l'on appelle les répertoires spécifiques.

data/

Contient tous les fichiers d'initialisation de la base de données de l'applicatif

```
[D] data/
 |-[F] .htaccess
 |-[D] pgsql/
    |---- ...
 |---- ...
```

app/

Contient tout les scripts spécifiques à l'applicatif que ce soit des javascripts ou des images ou des scripts PHP

```
[D] app/
 |-[D] css/
    |---- ...
 |-[D] img/
    |---- ...
 |-[D] js/
    |---- ...
 |---- ...
```

dyn/

Contient les fichiers de paramétrage de l'applicatif

```
[D] dyn
 |-[F] .htaccess
 |---- ...
```

gen/

Contient les scripts (obj) et requêtes (sql) générés par le générateur

```
[D] gen
 |-[F] .htaccess
 |-[D] dyn/
    |---- ...
 |-[D] inc/
    |---- ...
 |-[D] obj/
    |---- ...
 |-[D] sql/
    |-[D] pgsql
    |---- ...
 |---- ...
```

locales/

Contient les fichiers de traduction de l'applicatif

```
[D] locales/
 |-[F] .htaccess
 |---- ...
```

obj/

Contient les objets métiers surchargeant les objets générés

```
[D] obj/
 |-[F] .htaccess
 |---- ...
```

sql/

Contient les scripts sql surchargeant les scripts générés

```
[D] sql/
 |-[F] .htaccess
 |-[D] pgsql/
    |---- ...
 |---- ...
```

tests/

Contient les jeux de tests unitaires et fonctionnels de l'applicatif

```
[D] tests/
 |-[F] .htaccess
 |---- ...
```

tmp/

Contient les fichiers temporaires créés par l'applicatif

```
[D] tmp/
 |-[F] .htaccess
 |---- ...
```

trs/

Contient les fichiers stockés par l'applicatif

```
[D] trs/
 |-[F] .htaccess
 |-[D] 1/
    |---- ...
 |---- ...
```

2.1.1.2 Les répertoires du framework

Ces répertoires sont ceux qui sont issus du framework, c'est-à-dire qu'ils ne sont pas dans l'applicatif lui même. Par exemple, sur un applicatif comme openCimetière ce sont les répertoires qui vont être récupérés par une propriété externals sur le gestionnaire de versions que l'on appelle les répertoires du framework.

php/

Contient les librairies PHP utilisées par le framework (comme dbpear, phpmailer ou fpdf)

```
[D] php
 |-[F] .htaccess
 |---- ...
```

core/

Contient les classes de la librairie du framework

```
[D] core
 |-[F] .htaccess
 |---- ...
```

css/

Contient les feuilles de style de base du framework

```
[D] css/
 |---- ...
```

img/

Contient les images du framework

```
[D] img/
 |---- ...
```

js/

Contient les javascripts de base du framework

```
[D] js/
 |---- ...
```

lib/

Contient les librairies javascripts utilisées par le framework (comme openLayers ou jquery)

```
[D] lib/
 |---- ...
```

pdf/

Contient les scripts d'édition du framework

```
[D] pdf/
 |---- ...
```

scr/

Contient les scripts d'affichage du framework

```
[D] scr/
 |---- ...
```

spg/

Contient les sous programmes génériques du framework

```
[D] spg/
 |---- ...
```

2.1.2 Initialisation de la base de données

2.1.2.1 Description du dossier data/pgsql/

Il est nécessaire de positionner l'entête de fichier suivant pour chacun des fichiers sql de ce dossier :

```
--------------------------------------------------------------------------------
-- Description succinte de l'utilité du fichier
--
-- Informations nécessaires à la génération ou à la composition du fichier
--
-- @package <APPLICATIF>
-- @version SVN : $Id$
--------------------------------------------------------------------------------
```

Description de tous les fichiers init*.sql

Il s'avère nécessaire de mettre dans l'entête des fichiers `init*.sql` la commande ou les instructions qui ont permis de générer ou de composer le fichier en question.

2.1.2.1.1.1 Le fichier init.sql Ce fichier contient les instructions de base du framework. Il permet de créer les tables et les séquences du framework (celles qui commencent par *om_*). Il est généré grâce à la commande :

```
pg_dump -s -O -n <SCHEMA> -T <SCHEMA>.om_* <DATABASE>
```

Dans les applicatifs, ce fichier est sensé être directement copié depuis le framework.

2.1.2.1.1.2 Les fichiers `init_metier*.sql` Ces fichiers contiennent les instructions de base de l'applicatif.

– Le fichier `init_metier.sql` permet de modifier (si besoin) le modèle de données du framework et de créer les tables et les séquences de l'applicatif (celles qui ne commencent par *om_*). Il est généré grâce à la commande :

```
pg_dump -s -O -n <SCHEMA> -t <SCHEMA>.om_* <DATABASE>
```

Dans le framework, ce fichier est vide.

– Le fichier `init_metier_sig.sql` permet de modifier le modèle de données créé précédemment pour y ajouter des champs de type geom pour la gestion du SIG.

– Le fichier `init_metier_vue.sql` permet de modifier le modèle de données créé précédemment pour y remplacer une table par une vue vers la table d'un autre schéma ou d'une autre base de données.

2.1.2.1.1.3 Les fichiers `init_parametrage*.sql` Ces fichiers contiennent l'initialisation du paramétrage c'est-à-dire les données nécessaires à l'utilisation de l'application. Ils sont générés généralement grâce à la commande :

```
pg_dump -a -t <SCHEMA>.<TABLE1> -t <SCHEMA>.<TABLE2> ... <DATABASE>
```

– Le fichier `init_parametrage.sql` permet d'initialiser par exemple la ou les collectivités de base ainsi que les profils et l'utilisateur admin. Dans certains applicatifs simple, ce fichier peut être unique et tout le paramétrage contenu dans ce dernier.

– Le fichier `init_parametrage_permissions.sql` permet d'initialiser les permissions de l'applicatif. Cette initialisation se trouve dans un fichier séparé pour appréhender plus facilement le paramétrage des permissions et éventuellement la mise à jour de ce paramétrage.

– Le fichier `init_parametrage_editions.sql` permet d'initialiser les editions de base générique de l'aplicatif. Cette initialisation se trouve dans un fichier séparé pour appréhender plus facilement le paramétrage des éditions et éventuellement la mise à jour de ce paramétrage.

– Le fichier `init_parametrage_*.sql` peut permettre de découper encore l'initialisation pour appréhender plus facilement le paramétrage et éventuellement la mise à jour de ce paramétrage.

2.1.2.1.1.4 Le fichier `init_data.sql` Ce fichier contient l'initialisation d'un jeu de données à destination de trois environnements distincts :

– l'environnement de développement,

– l'environnement de démonstration,

– l'environnement de tests.

Description des fichiers `vX.X.X.sql` ou `ver_X.X.X.sql`

Ces fichiers permettent de mettre à jour les applicatifs d'une version vers la version supérieure. Le X.X.X correspond au numéro de version vers lequel la mise à jour se fait et depuis la version juste précédente.

Lorsque le framework ou l'applicatif est en développement, ce fichier peut être suffixé par *-dev* et indique qu'il n'a pas encore été intégré aux différents fichiers `init*.sql`. Juste avant une nouvelle version du framework, les fichiers `init*.sql` doivent être regénérés pour intégrés les dernières modifications et ce fichier renommé avec son nom `vX.X.X.sql` ou `ver_X.X.X.sql` standard.

Description du fichier `update_sequences.sql`

Ce fichier permet de créer une fonction capable de mettre à jour toutes les séquences correctement liées aux champs auxquels elles se rattachent en fonction de la dernière valeur du champ dans la table. En plus de la création de la fonction ce script exécute la fonction.

Description du fichier `install.sql`

Ce fichier permet d'initialiser tous les fichiers qui sont décrits ci-dessus dans le bon ordre. Par défaut ce fichier installe la base de données et les données nécessaires aux trois environnements suivants :
– l'environnement de développement,
– l'environnement de démonstration,
– l'environnement de tests.

Note : Ce fichier comporte l'initialisation des commandes postgis par défaut pour la dernière version de postgis. Les commandes pour l'ancienne version sont présentes et commentées dans ce même fichier.

2.1.2.2 Paramétrage de la connexion à la base de données

Le paramétrage de la connexion à la base de données se fait dans le fichier `dyn/database.inc.php`.

Note : Dans le framework le schéma utilisé par défaut est *openexemple*, dans les applicatifs c'est normalement le nom de l'applicatif *<APPLICATIF>* (par exemple : *openelec*, *opencimetiere*, ...).

```
<?php
/**
 * Ce fichier permet le paramétrage de la connexion à la base de données,
 * chaque entrée du tableau correspond à une base différente. Attention
 * l'index du tableau conn représente l'identifiant du dossier dans lequel
 * seront stockés les fichiers propres a cette base dans l'application.
 *
 * @package openmairie_exemple
 * @version SVN : $Id: database.inc.php 2302 2013-05-23 18:04:22Z fmichon $
 */

// PostGreSQL
$conn[1] = array(
    "openExemple", // Titre
    "pgsql", // Type de base
    "pgsql", // Type de base
    "postgres", // Login
    "postgres", // Mot de passe
    "tcp", // Protocole de connexion
    "localhost", // Nom d'hote
    "5432", // Port du serveur
    "", // Socket
    "openexemple", // Nom de la base
    "AAAA-MM-JJ", // Format de la date
    "openexemple", // Nom du schéma
    "", // Préfixe
    NULL, // Paramétrage pour l'annuaire LDAP
    "mail-default", // Paramétarge pour le serveur de mail
    "filestorage-default", // Paramétrage pour le stockage des fichiers
);

?>
```

La documentation de DB PEAR qui est le module d'abstraction utilisé par le framework donne plus d'informations sur les paramètres.

2.1.3 Paramétrage du framework

Le paramétrage de l'application se fait dans le répertoire /dyn.

Il est proposé dans ce chapitre de décrire les différents fichiers de paramétrage.

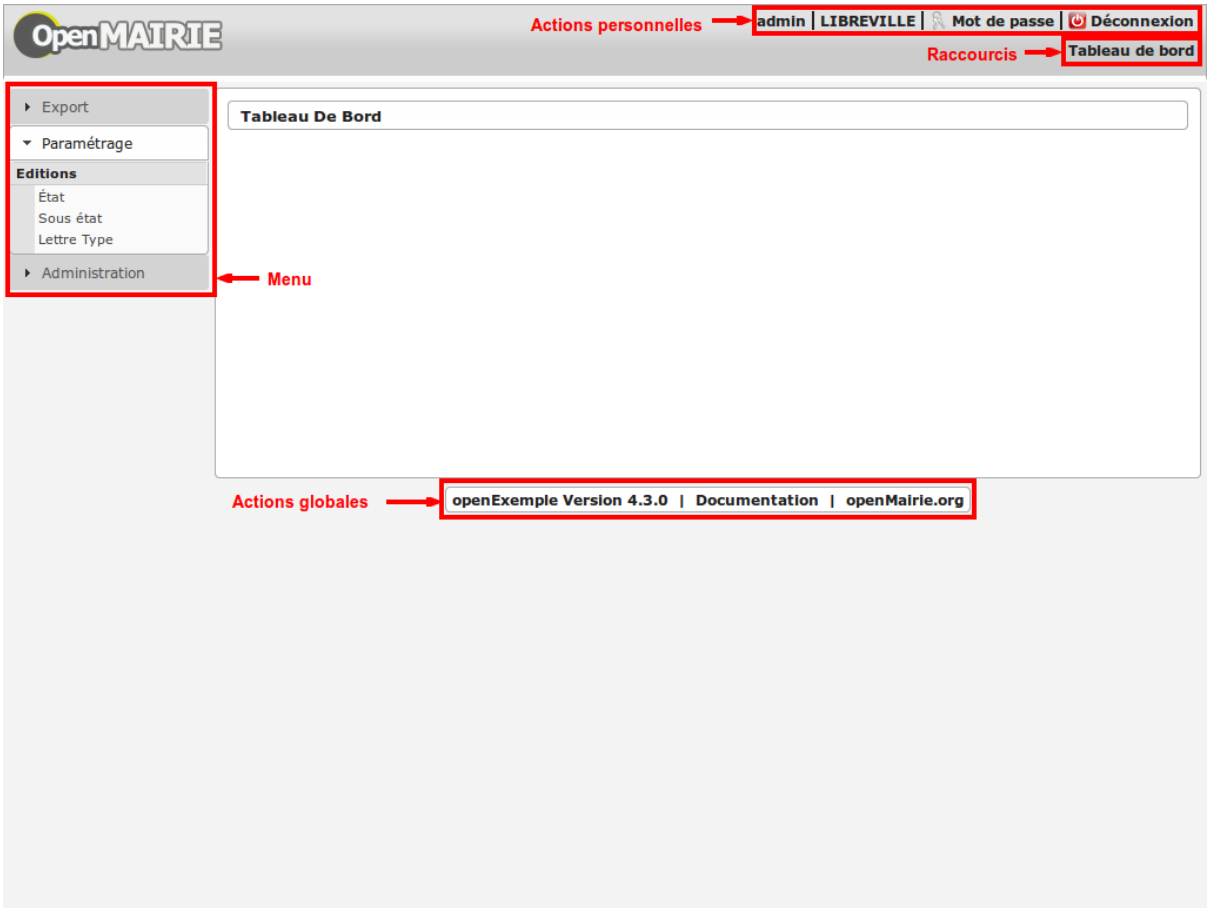
Pour le fichier dyn/database.inc.php, voir *Paramétrage de la connexion à la base de données*.

Les fichiers de paramétrage sont les suivants

```
dyn/menu.inc.php               menu principal à gauche
dyn/action.inc                 menu haut
dyn/shortslink.inc             lien sous menu haut
dyn/dashboard.inc.php          tableau de bord
dyn/locales.inc                application
dyn/config.inc.php             application
dyn/include.inc.php            chemin d'accès aux librairies
dyn/debug.inc.php              mode debug
dyn/version.inc                paramétrage de la version
dyn/var_sig.inc                paramétrage sig
dyn/form_sig_update.inc.php    parametrage sig
dyn/form_sig_delete.inc.php    parametrage sig
dyn/var_adresse_postale.inc    parametrage sig
dyn/filestorage.inc.php        filestorage
dyn/footer.inc.php             enpieds
dyn/tab.inc.php                variable specifique à passer dans l'url pour tab.php
dyn/soustab.php                variable specifique à passer dans l'url pour soustab.php
dyn/form.php                   variable specifique à passer dans l'url pour form.php
dyn/sousform.php               variable specifique à passer dans l'url pour sous.php
dyn/var.inc                    variable application (deprecated : préférez om_parametre
dyn/varetatpdf.inc             variable etat et sousetat pdf
dyn/varlettretypepdf.inc       varaible lettre type
dyn/varsousform.get.specific.inc.php    variable spécifique sousform
dyn/directory.inc.php          connexions a des annuaires LDAP
dyn/comboaffichage.inc.php     paramétrage combo
dyn/comboparametre.inc.php     parametrage combo
dyn/comboretour.inc.php        paramétrage combo


README.txt                     fichiers textes
HISTORY.txt
LICENCE.txt
TODO.txt
INSTALL.txt

app/SPECIFIC.txt                explication sur la partie spécifique de l application
```

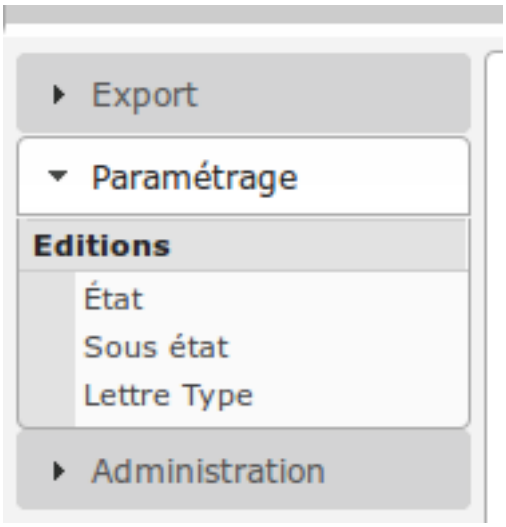
2.1.3.1 Les zones de navigation

Quatre zones de navigation différentes sont disponibles dans le framework :



Le menu

Le but de cette zone de navigation est de rassembler les liens vers toutes les fonctions du logiciel regroupées par rubrique et catégorie. Elle se situe à gauche du contenu et est visible uniquement lorsque l'utilisateur est authentifié.



Par défaut **le menu** est composé de la manière suivante

```
application             vide par défaut, contient l'accès à votre application
export                  contient le script "edition" qui reprend
                            les éditions pdf des tables
                        contient le menu "reqmo" qui reprend les requêtes
```

```
                        mémorisées
traitement              vide par défaut, cet option contient les scripts de
                        traitements
parametrage             Cette option contient vos tables de paramétrage
                        fonctionnel. Par défaut il contient le
                        paramétrage des états / sous-états / lettres type
administration          Cette option contient les fonctions de configuration
                        de l'administrateur technique. Cela comprend
                        notamment le paramètrage de la collectivité,
                        om_sig  et la gestion des droits d'accès
```

La configuration des liens se fait dans le fichier `dyn/menu.inc.php`. Ce fichier de paramétrage n'est pas obligatoire. Si il n'existe pas, aucun lien n'est affiché. Ce fichier de paramétrage doit contenir la déclaration d'un tableau de tableaux associatifs dans la variable `$menu`. Chaque tableau associatif représente une rubrique. Chaque rubrique contient un tableau de tableaux associatifs, chacun représentant un lien.

Les caracteristiques de ce tableau sont les suivantes :

tableau rubrik

```
title (obligatoire)
description (texte qui s'affiche au survol de la rubrique)
href (contenu du lien href)
class (classe css qui s'affiche sur la rubrique)
right (droit que l'utilisateur doit avoir pour visionner cette rubrique)
links (obligatoire)
open (critères de pré-ouverture de cette rubrique du menu)
```

tableau links

```
title (obligatoire)
href (obligatoire) (contenu du lien href)
class (classe css qui s'affiche sur l'element)
right (droit que l'utilisateur doit avoir pour visionner cet element)
target (pour ouvrir le lien dans une nouvelle fenetre)
open (critères de pré-ouverture de la rubrique du menu dans laquelle est ce
      lien, et sélection de ce lien en lien actif)
```

L'entrée `open` sert à marquer une entrée de menu comme active. La rubrique contenant cette entrée est ouverte dès l'affichage de la page, et l'entrée active est mise en évidence. L'entrée `open` peut contenir soit une chaîne soit un `array()` comportant plusieurs chaînes. Chaque chaîne est créée selon la syntaxe `'script.php|obj'`, chacune des deux parties étant optionnelle. Le caractère séparateur `|` est obligatoire.

Exemple [`'\|om_collectivite'` sélectionnera l'entrée pour toutes les url] ayant `obj=om_collectivite`

`'tab.php|om_collectivite'` sélectionnera l'entrée pour l'affichage du tableau de la classe `om_collectivite`

`'unecran.php|'` sélectionnera l'entrée dès lors que le script `unecran.php` est appelé quelque soit la classe `obj`

Les actions personnelles

Le but de cette zone de navigation est de regrouper des liens vers des fonctions qui concernent les informations de connexion de l'utilisateur. Elle se situe dans le coin en haut à droite de l'écran et est visible uniquement lorsque l'utilisateur est authentifié.

Par défaut **les actions personnelles** sont composées de quatre éléments :
– le login de l'utilisateur,
– le libellé de la collectivité,
– un lien vers la page de modification du mot de passe,
– un lien vers la page de déconnexion du logiciel.
Le login de l'utilisateur est récupéré par la méthode `displayActionLogin()` de la classe `om_application`. Cette méthode peut être surchargée dans la classe `utils`.

Le libellé de la collectivité est récupéré par la méthode `displayActionCollectivite()` de la classe `om_application`. Cette méthode peut être surchargée dans la classe `utils`.

La configuration des liens se fait dans le fichier `dyn/actions.inc.php`. Ce fichier de paramétrage n'est pas obligatoire. Si il n'existe pas, aucun lien n'est affiché. Ce fichier de paramétrage doit contenir la déclaration d'un tableau de tableaux associatifs dans la variable `$actions`. Chaque tableau associatif représente un lien.

```
<?php
//
$actions = array();
//
$actions[] = array(
    "title" => _("Link"),
    "description" => _("Description"),
    "href" => "../scr/link.php",
    "target" => "_blank",
    "class" => "action-link",
    "right" => "link",
);
?>
```

Description de chaque paramètre du tableau associatif :

Paramètre	Requis ?	Description
title	O	Texte
description	N	Texte qui s'affiche au survol de l'élément
href	N	Contenu du lien href
target	N	Attribut pour ouvrir le lien dans une nouvelle fenêtre
class	N	Classe CSS qui s'affiche sur l'élément
right	N	Permission nécessaire à l'utilisateur pour visualiser l'élément

Les raccourcis

Le but de cette zone de navigation est de regrouper des liens vers des fonctions précises utilisées très souvent. Elle se situe en haut à droite de l'écran juste au dessous des actions personnelles et est visible uniquement lorsque l'utilisateur est authentifié.

Par défaut **les raccourcis** contiennent uniquement un lien vers le tableau de bord.

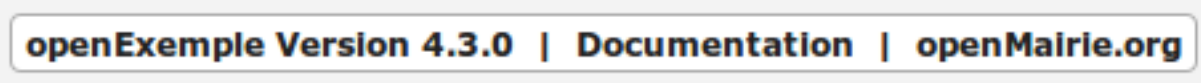
La configuration des liens se fait dans le fichier `dyn/shortlinks.inc.php`. Ce fichier de paramétrage n'est pas obligatoire. Si il n'existe pas, aucun lien n'est affiché. Ce fichier de paramétrage doit contenir la déclaration d'un tableau de tableaux associatifs dans la variable `$shortlinks`. Chaque tableau associatif représente un lien.

```
<?php
// On initialise le tableau conteneur
$shortlinks = array();
// On ajoute au tableau conteneur un tableau associatif représentant un lien
// (à répéter autant de fois que nécessaire)
$shortlinks[] = array(
    "title" => _("Link"),
    "description" => _("Description"),
    "href" => "../scr/link.php",
    "target" => "_blank",
    "class" => "action-link",
    "right" => "link",
);
?>
```

Paramètre	Requis ?	Description
title	O	Texte
description	N	Texte qui s'affiche au survol de l'élément
href	N	Contenu du lien href
target	N	Attribut pour ouvrir le lien dans une nouvelle fenêtre
class	N	Classe CSS qui s'affiche sur l'élément
right	N	Permission nécessaire à l'utilisateur pour visualiser l'élément

Les actions globales

Le but de cette zone de navigation est de représenter la section "À propos" du logiciel. Elle se situe en bas de l'écran juste au dessous du contenu de la page et est visible lorsque l'utilisateur est authentifié ou non.



Par défaut **les actions globales** sont composées de trois éléments :
– le nom du logiciel ainsi que son numéro de version,
– un lien vers la documentation du site openMairie,
– un lien vers le site openMairie.
Le nom du logiciel est récupéré de la variable `$config['application']` présente dans le fichier `dyn/config.inc.php`. La version est récupérée de la variable `$version` présente dans le fichier `dyn/version.inc.php`.

La configuration des liens se fait dans le fichier `dyn/footer.inc.php`. Ce fichier de paramétrage n'est pas obligatoire. Si il n'existe pas, aucun lien n'est affiché. Ce fichier de paramétrage doit contenir la déclaration d'un tableau de tableaux associatifs dans la variable `$footer`. Chaque tableau associatif représente un lien.

```
<?php
// On initialise le tableau conteneur
$footer = array();
// On ajoute au tableau conteneur un tableau associatif représentant un lien
// (à répéter autant de fois que nécessaire)
$footer[] = array(
    "title" => _("Link"),
    "description" => _("Description"),
    "href" => "../scr/link.php",
    "target" => "_blank",
    "class" => "action-link",
    "right" => "link",
);
?>
```

Paramètre	Requis ?	Description
title	O	Texte
description	N	Texte qui s'affiche au survol de l'élément
href	N	Contenu du lien href
target	N	Attribut pour ouvrir le lien dans une nouvelle fenêtre
class	N	Classe CSS qui s'affiche sur l'élément
right	N	Permission nécessaire à l'utilisateur pour visualiser l'élément

2.1.3.2 Le tableau de bord

Le tableau de bord se paramètre dans le fichier *dyn/dashboard.inc*.

Ce fichier est appellé par le script scr/dashboard.php.

Pour avoir son propre tableau de bord, il suffit de decommenter la ligne // die() ; et on accède plus au widget

Voir chapitre : widget et tableau de bord paramétrable

2.1.3.3 Les variables locales et la langue

Les variables locales sont paramétrées dans le fichier *dyn/locales.inc.php*

Ce fichier contient :
– le paramétrage du codage des caracteres (ISO-8859-1 ou UTF8)

```
"DEPRECATED"

    define('CHARSET', 'ISO-8859-1');
    ou
    define('CHARSET', 'UTF8');

Dans la version 4.2.0, il y a 2 paramètres :

    pour la base : DB_CHARSET
    pour apache  : HTTP_CHARSET

    Ces 2 paramètres remplacent CHARSET


Note ::

    Dans apache, il est possible de modifiet l'encodage
    dans etc/apache2/apache2.conf commenter ##AddDefaultCharset = ISO-8859-1
    relancer ensuite apache : $ etc/apache2/init.d/apache2 reload

    A partir de la version 3.0.1, l'imcompatibilité utf8 de la bibliotheque fpdf est traitée
```

– le dossier ou sont installées les variables du systeme

```
define('LOCALE', 'fr_FR');
```

– Le dossier contenant les locales et les fichiers de traduction

```
define('LOCALES_DIRECTORY', '../locales');
```

– Le domaine de traduction

```
define('DOMAIN', 'openmairie');
```

Les zones à traduire sont sous le format : _("zone a traduire")

Voir le chapitre sur les outils : *poEdit*

2.1.3.4 Le paramétrage de l application metier

L'application métier est paramétrée dans *dyn/var.inc*

Ce script contient les paramétres globaux de l application . Attention les paramètres s'appliquent à toutes les bases de l'application.

Le paramétrage spécifique par collectivité doit se faire dans la table om_parametre

La configuration générale de l'application se fait aussi dans *dyn/config.inc.php*.

Les paramètres sont récupérés avec la création d'un objet utils par : $f->config['nom_du_parametre']

Voir framework/utilitaire

Exemple de paramétrage avec openCourrier

```
$config['application'] = _("openCourrier");
$config['title'] = ":: "._("openMairie")." :: "._("openCourrier");
$config['session_name'] = "openCourrier";
```

– le mode demonstration de l'application se paramétre avec $config['demo']

Ce mode permet de pre-remplir le formulaire de login avec l'identifiant 'demo' et le mot de passe 'demo'

```
$config['demo'] = false;  l'application n'est pas en mode démo
                  true; l'application est en mode démo

Attention, pour empêcher de changer le mot de passe, il faut paramétrer l'accès
dans la table om_droit : password
```

– La configuration des extensions autorisees dans le module upload.php

Pour changer votre configuration, décommenter la ligne et modifier les extensions avec des " ;" comme séparateur

```
$config['upload_extension'] = ".gif;.jpg;.jpeg;.png;.txt;.pdf;.csv;"
```

– La configuration de la taille maximale des fichiers dans le module upload.php

Pour changer votre configuration, décommenter la ligne et modifier la taille. La taille maximale est en mo.

```
$config['upload_taille_max'] = str_replace('M', '', ini_get('upload_max_filesize')) * 1024;
```

– Le thème de l'application

A partir de la version 3.1.0, le theme n'est plus géré dans config.inc.php. Il est initialisé dans EXTERNALS.TXT du repertoire om-theme (version 4.2.0)

```
exemple pour om_ui_darkness

om_theme svn://scm.adullact.net/svnroot/openmairie/externals/jquery-ui-theme/
                om_ui-darkness/tags/1.8.14
```

2.1.3.5 Le Parametrage des librairies

Le paramétrage de l'accès aux librairies se fait dans *dyn/include.inc.php*

> Ce fichier permet de configurer les paths en fonction de la directive include_path du fichier php.ini. Vous pouvez aussi modifier ces chemins avec vos propres valeurs si vous voulez personnaliser votre installation :
>
> PEAR
>
> ```
> array_push($include, getcwd()."/../php/pear");
> ```
>
> DB

```
array_push($include, getcwd()."/../php/db");
```

FPDF

```
array_push($include, getcwd()."/../php/fpdf");
```

OPENMAIRIE (dans CORE depuis la version 4.2.0)

```
define("PATH_OPENMAIRIE", getcwd()."../core/openmairie/");
```

Par défaut, les librairies sont incluses dans openmairie_exemple :
– /lib : contient les librairies javascript
– /php : contient les librairies php

2.1.3.6 Le mode DEBUG

Dans le code, pour logger une information, il suffit de d'utiliser la ligne suivante :

```
$this->addToLog("requete sig_interne maj parcelle inexistante :".$sql, EXTRA_VERBOSE_MODE);
```

Les différents modes DEBUG présents dans l'application sont définis dans le fichier *core/om_debug.inc.php* :
– EXTRA_VERBOSE_MODE - mode "très bavard" : affiche tous les messages
– VERBOSE_MODE - mode "bavard" : affiche tous les messages d'erreur ainsi que toutes les requêtes exécutées
– DEBUG_MODE - mode "debug" : affiche tous les messages d'erreur
– PRODUCTION_MODE - mode "production" : il n y a pas de message
Dans l'applicatif, on peut paramétrer le mode DEBUG grâce au fichier *dyn/debug.inc.php* suivant. Il suffit de commenter/décommenter l'instruction define souhaitée.

```
<?php
/**
 * Ce fichier contient le parametrage pour le mode debug
 *
 * @package openmairie_exemple
 * @version SVN : $Id: debug.inc.php 2198 2013-03-28 17:08:33Z fmichon $
 */

/**
 *
 */
(defined("PATH_OPENMAIRIE") ? "" : define("PATH_OPENMAIRIE", ""));
require_once PATH_OPENMAIRIE."om_debug.inc.php";

/**
 *
 */
//define('DEBUG', EXTRA_VERBOSE_MODE);
//define('DEBUG', VERBOSE_MODE);
//define('DEBUG', DEBUG_MODE);
define('DEBUG', PRODUCTION_MODE);

?>
```

Dans le dossier tmp/error_log.txt les messages de logs de niveau DEBUG_MODE sont écrits quelque soit le mode définit.

2.1.3.7 La version de votre application

Vous devez mettre le numéro de version et la date de votre application dans *dyn/version.inc*

Voir *le versionage des applications*.

2.1.3.8 Les informations generales

Les fichiers textes d'information générale sont à la racine de l'application :

README.txt :

ce fichier peut contenir entre autre, la liste des auteurs ayant participé au projet

HISTORY.txt : information sur chaque version :

les (+) et les (bugs) corrigés

app/SPECIFIC.txt :

Ici, vous décrivez la specificite de l application courante par rapport au framework

LICENCE.txt : licence libre de l application

TODO.txt : feuille de route - roadmap

INSTALL.txt : installation de l application

2.1.3.9 L'installation automatique

Lun fichier data/sql/install.sql permet d'installer rapidemment et data/sql/make_init.sh permet de constituer rapidemment des scripts sql d'installation.

2.1.3.10 Les paramétres des combos

Les paramétres des combos sont paramétrés dans les fichiers suivants (type de contrôle de formulaire comboD et comboG (pour formulaire) ou comboD2 et comboG2 (pour sous formulaire)

```
- comboaffichage.inc.php :
    paramétre de l'affichage dans la fenêtre combo.php
- comboparametre.inc.php
    affecte des valeus spécifiques au formulaire parent si il y a plusieurs
    enregistrement en lien (choix en affichage)
- comboretour.inc.php
    meme chose que comboparametre.inc si il n'y a qu un enregistrement en lien
    (pas d'affichage de la fenetre)
```

Voir *chapitre framework/formulaire, sous programme générique combo.php*

2.1.3.11 Les paramétres éditions

Les variables dans les éditions sont paramétrées dans

```
- varpdf.inc                pour les pdf
- varetatpdf.inc            pour les états et les sous états
- varlettretypepdf.inc      pour les lettres type
```

Voir *chapitre framework/édition*

2.1.3.12 Les paramétres om_sig

var_sig.php

les paramètres sont les suivants

```
$contenu_etendue[0]= array('4.5868,43.6518,4.6738,43.7018'
                           );
$contenu_etendue[1]= array('vitrolles'
                           );
$contenu_epsg[0] = array("","EPSG:2154","EPSG:27563");
$contenu_epsg[1] = array("choisir la projection",'lambert93','lambertSud');
$type_geometrie[0] = array("","point","line","polygon");
$type_geometrie[1] = array("choisir le type de géométrie",'point','ligne','polygone');
```

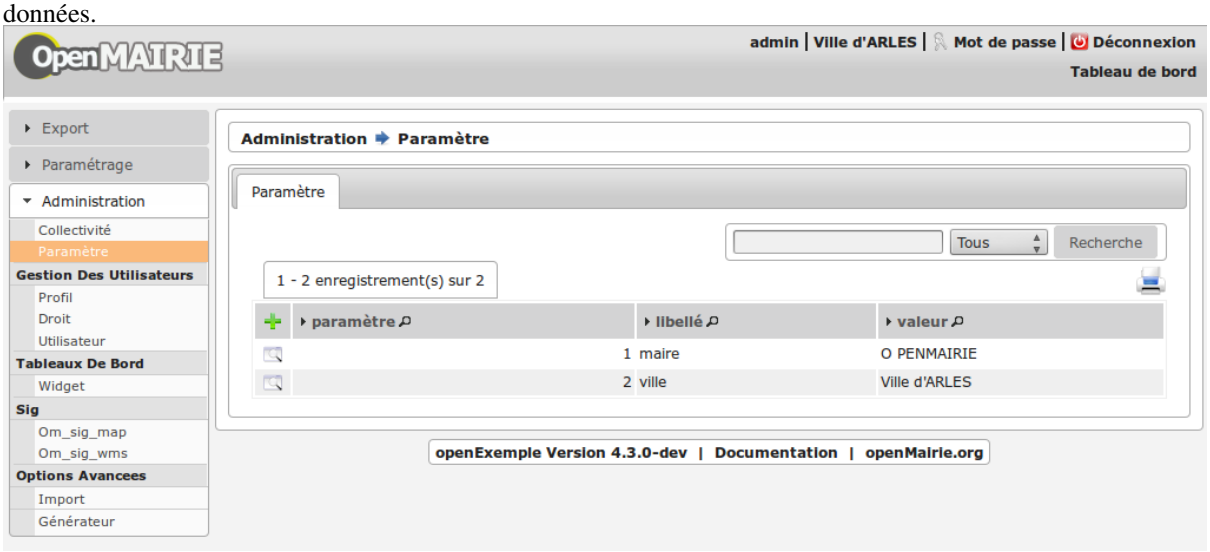
ces paramétres sont utilisés pour la saisie de carte : voir chapitre sig

Les post traitements de form_sig permettent de faire des traitement apres saisie de géométries avec om_sig

form_sig_update.inc.php
form_sig_delete.inc.php

2.1.4 Afficher les tables

Il est décrit dans ce paragraphe, l'utilisation et la configuration des tableaux d'enregistrements issus de la base de données.



La gestion des tableaux se base sur le fichier core/om_table.class.php (classe table)

Pour chaque affichage de tableau, le script sql/DBTYPE/[objet].inc.php correspondant est appelé. Il permet de stocker le détail des requêtes nécessaires à l'affichage du contenu.

Les tableaux sont construits lors de l'appel aux scripts scr/tab.php et scr/soustab.php.

2.1.4.1 Les script scr/tab.php et scr/soustab.php

L'appel à ces scripts permet d'afficher un tableau d'enregistrements de l'objet passé en paramètre.

Liste des paramètres passés à l'url :
– obj : nom de l'objet pour lequel on souhaite afficher le tableau
– premier : numéro de la première ligne affichée dans le tableau
– recherche : chaîne de caractères recherchée depuis le modules de recherche
– selectioncol : numéro de la colonne sélectionnée dans le module de recherche
– tricol : numéro de colonne et orientation (+/-) du tri du tableau
– valide : (true/false) affiche ou non les enregistrements non valide
L'appel à scr/soustab.php est fait en javascript depuis un formulaire afin d'afficher les informations liées à l'enregistrement en cours d'édition.

2.1.4.2 La requête SQL d'affichage

Elle se trouve dans sql/DBTYPE/[objet].inc.php

Les paramètres sont les suivants pour om_parametre.inc.php

```
<?php
//Nombre d'enregistrements par page
$serie=15;
//Icone affiché (XXX à voir deprecated)
$ico="../img/ico_application.png";
//Titre du tableau
$ent = _("option")." -> "._("om_parametre");
//Table de référence (il peut y avoir une ou plusieurs jointure)
$table=DB_PREFIXE."om_parametre";
//Liste des champs du tableau
$champAffiche=array('om_parametre',
                    'libelle',
                    'valeur',
                    'om_collectivite');
//Champs pour la recherche
$champRecherche=array('libelle','valeur');
//Critère de tri par défaut
$tri="";
//édition PDF
$edition="om_parametre";
//sous formulaire(s) associé(s)
$sousformulaire= array()

//autre exemple de sous-formulaire avec om_collectivite.inc.php
$sousformulaire=array('om_etat',
                'om_lettretype',
                'om_parametre',
                'om_sousetat',
                'om_utilisateur');
?>
```

Il est possible de surcharger les liens du tableau (voir *la configuration des actions*)

2.1.4.3 Le composant openMairie

tab.php utilise les méthodes d'om_table.class.php qui est une classe d'openMairie

`core/om_table.class.php`

Les méthodes de ce composant peuvent être surchargées dans obj/om_table.class.php

2.1.5 La recherche avancée

2.1.5.1 Les différents types de recherche

Recherche simple

Cette recherche est celle disponible par défaut sur les tableaux d'openMairie.

Elle permet de :
– rechercher une valeur dans une colonne parmi toutes celles affichées ;
– rechercher une valeur dans toutes les colonnes affichées ;
– rechercher des valeurs approximatives.

Note : Il est possible de modifier la liste des colonnes dans laquelle est effectuée la recherche. Cette liste ne correspond pas forcément aux colonnes affichées. Elle correspond seulement par défaut, c'est à dire lorsqu'aucune surcharge ne modifie les fichiers générés dans `gen/sql/`.

Recherche avancée

Cette recherche est une fonctionnalité qui peut être activée et configurée manuellement pour un ou plusieurs tableaux donnés.

Elle permet de :
– afficher un formulaire de recherche mono-critère permettant d'effectuer des recherches strictes ou approximatives ;
– afficher un formulaire de recherche multi-critères permettant d'effectuer des recherches strictes ou approximatives ;
– rechercher des valeurs dans des tables et des colonnes qui ne sont pas affichées.

2.1.5.1.2.1 Recherche avancée mono-critère Le formulaire de recherche mono-critère est un formulaire ne s'affichant que si la recherche avancée est activée. Il permet aux utilisateurs de basculer sur un formulaire similaire à celui de recherche simple lorsque la recherche avancée est activée.

Ce formulaire se comporte de la même manière que celui de recherche simple, avec quelques différences :
– il permet de rechercher des valeurs strictes ou approximatives (par défaut approximatives) ;
– il recherche dans toutes les colonnes proposées par la recherche simple ;
– il conserve les valeurs recherchées après la réalisation d'une action (ajout, modification, etc...) ;
– il dispose d'un bouton `Vider le formulaire` permettant de vider les champs ;
– il dispose d'un bouton + permettant de basculer sur le formulaire multi-critères.

2.1.5.1.2.2 Recherche avancée multi-critères Le formulaire de recherche multi-critères est un formulaire ne s'affichant que si la recherche avancée est activée. Il permet aux utilisateurs de bénéficier de plusieurs champs, et ainsi effectuer des recherches plus précise qu'avec le formulaire de recherche simple.

Description du formulaire :
– il peut afficher plusieurs champs, de type texte, nombre, date ou liste à choix ;
– il permet, pour chaque tableau, de configurer la liste des champs affichés ;
– il permet, pour chaque champ, de rechercher des valeurs strictes ou approximatives (par défaut approximatives) ;
– il permet, pour chaque champ, de rechercher des valeurs dans des tables et des colonnes qui ne sont pas affichées ;
– il conserve les valeurs recherchées après la réalisation d'une action (ajout, modification, etc...) ;
– il dispose d'un bouton `Vider le formulaire` permettant de vider les champs ;
– il dispose d'un bouton + permettant de basculer sur le formulaire mono-critère.

2.1.5.2 Configuration de la recherche avancée

Activation

Exemple avec le modèle `om_utilisateur`.

Pour activer la recherche avancée, rendez-vous dans le fichier `sql/sgbd/om_utilisateur.inc.php` et ajoutez la configuration suivante au tableau d'options :

```
<?php

 $options[] =  array('type' => 'search',
                     'display' => true,
                     'advanced' => $champs,
                     'default_form' => 'advanced',
```

```
                    'absolute_object' => 'om_utilisateur');

?>
```

Note : A partir de la version 4.3.0 d'openMairie, le tableau `$options` est disponible dans les fichiers `sql/` de l'application. Il n'est plus nécessaire de le déclarer manuellement.

La clé `type` est obligatoire. Elle permet de définir le type de l'option. Pour une recherche il faut saisir `search`.

La clé `display` est obligatoire. Elle permet d'afficher ou non la recherche, tout en conservant sa configuration.
– `true` permet d'afficher la recherche ;
– `false` permet de masquer la recherche.
La clé `advanced` est obligatoire (pour la recherche avancée). Elle permet de préciser que le formulaire de recherche est un formulaire de recherche avancée et non simple. Cette clé doit contenir le tableau des champs configurés pour la recherche (voir plus bas pour la configuration des champs).

La clé `default_form` est optionnelle. Elle permet de choisir quel formulaire de recherche est ouvert par défaut. La valeur `advanced'` permet d'afficher le formulaire multi-critères. Les autres valeurs, ou si `default_form` n'est pas configuré, affichent le formulaire mono-critère.

La clé `absolute_object` est obligatoire. Elle permet de spécifier à openMairie le nom du modèle l'objet recherché. Ce nom est celui du fichier dans `obj/`, ici `om_utilisateur.class.php` (sans son extension).

Autres paramètres

Wildcard

Le wildcard permet de rendre la recherche stricte ou approximative.

Cette option peut se configurer pour un ou plusieurs modèles particuliers dans les fichiers correspondants du répertoire `sql/` de l'application. Elle peut également être configurée de manière globale pour l'ensemble dans modèle à partir du fichier `dyn/tab.inc.php`.

Par défaut, il est paramétré de la manière suivante :

```
<?php

$options[] = array('type' => 'wildcard', 'left' => '%', 'right' => '%');

?>
```

– `left` détermine, dans la requête SQL de recherche, le caractère ajouté au début (à gauche) de la valeur recherchée ;
– `right` détermine, dans la requête SQL de recherche, le caractère ajouté en fin (à droite) de la valeur recherchée.

Avec cette configuration lorsque le mot « admin » est recherché dans une colonne, toutes les valeurs contenant « admin » sont retournées.

En modifiant la configuration de cette manière :

```
<?php

$options[] = array('type' => 'wildcard', 'left' => '', 'right' => '%');

?>
```

Seules les valeurs **commençant** par « admin » seront retournées.

Enfin avec :

```
<?php

$options[] = array('type' => 'wildcard', 'left' => '', 'right' => '');

?>
```

Seules les valeurs égales **exactement** à « admin » seront retournées.

2.1.5.3 Configuration des critères de recherche

La recherche avancée ne fonctionnera pas tant que la liste des champs du formulaire multi-critères n'aura pas été créée. Ces champs sont appelés ici des critères de recherche.

Configuration simple

Un critère de recherche est représenté par un tableau PHP contenant sa configuration.

```
<?php

$champs['identifiant_utilisateur'] =
    array('colonne' => 'om_utilisateur',
          'table' => 'om_utilisateur',
          'type' => 'text',
          'libelle' => _('Identifiant'),
          'taille' => 10,
          'max' => 8));

?>
```

La clé `identifiant_utilisateur` est le nom du champ HTML qui sera affiché sur le formulaire.

La clé `colonne` est obligatoire. Elle contient le nom de la colonne de la base de données qui sera interrogée si la variable `$_POST` contient la clé `identifiant_utilisateur`.

La clé `table` est obligatoire. Elle contient le nom de la table de la base de données qui sera interrogée si la variable `$_POST` contient la clé `identifiant_utilisateur`.

La clé `'type` est obligatoire. Elle contient le type du champ HTML à afficher. Cela peut être `date`, `text`, `select`, ou tout autre méthode de la classe `formulaire`. Pour les champs de type `select`, le nom du champ HTML doit être le même que le nom de la colonne.

La clé `libelle` est obligatoire. Elle contient le libellé qui sera affiché à côté du champ dans le formulaire de recherche.

La clé `taille` est optionnelle. Elle contient la taille du champ HTML (attribut HTML `size`).

La clé `max` est optionnelle. Elle contient la longueur maximale de la valeur du champ HTML (attribut HTML `maxlength`).

Une fois tous les critères de recherche configurés, il faudra simplement vérifier que le tableau des critères est bien utilisé par l'option de type `search`.

Exemple de formulaire pour le tableau du modèle `om_utilisateur` :

```
<?php

$champs = array();

$champs['login'] = array(
    'table' => 'om_utilisateur',
    'colonne' => 'login',
    'type' => 'text',
    'libelle' => _('Login'));
```

```
$champs['email'] = array(
    'table' => 'om_utilisateur',
    'colonne' => 'email',
    'type' => 'text',
    'libelle' => _('E-mail'));

$champs['om_profil'] = array(
    'table' => 'om_utilisateur',
    'colonne' => 'om_profil',
    'type' => 'select',
    'libelle' => _('Profil'));

 $options[] =  array('type' => 'search',
                     'display' => true,
                     'advanced' => $champs,
                     'default_form' => 'advanced',
                     'absolute_object' => 'om_utilisateur');

?>
```

Configuration avancée

2.1.5.3.2.1 Créer un intervalle de date Exemple : recherche des utilisateurs crées entre telle et telle date.

```
<?php

$champs['date_de_creation'] =
    array('colonne' => 'creation_date',
          'table' => 'user',
          'libelle' => _('Date de creation'),
          'type' => 'date',
          'where' => 'intervaldate');

?>
```

Cette configuration permet de créer deux champs HTML `datepicker` :
– `date_de_creation_min` : permettra de saisir une date minimale
– `date_de_creation_max` : permettra de saisir une date maximale
Ces champs permettent de rechercher les utilisateurs dont la date de créations est incluse dans l'intervalle saisi, bornes comprises. Il est possible de ne saisir qu'une seule date afin de rechercher les utilisateurs ayant été créés avant ou après une date particulière.

2.1.5.3.2.2 Créer un champ de recherche avec menu déroulant personnalisé Exemple : recherche des utilisateurs administrateurs.

Dans cet exemple, l'information se trouve directement dans la table interrogée.

```
<?php

// soit 'user' une table contenant une colonne 'is_admin'

$args = array();
$args[0] = array('', 'true', 'false');
$args[1] = array(_('Tous'), _('Oui'), _('Non'));

$champs['administrator'] =
    array('colonne' => 'is_admin',
          'table' => 'user',
          'libelle' => _('Administrateur'),
          'type' => 'select',
```

```
        'subtype' => 'manualselect',
        'args' => $args);

?>
```

Cette configuration permet de créer un champ HTML de type `select` avec trois choix :
– Tous (valeur '') ;
– Oui (valeur `true`) ;
– Non (valeur `false`).
Le tableau `$args[0]` contient les valeurs associées aux choix. Elles seront recherchées telles quelles dans la base de données.

En sélectionnant « Oui », la requête SQL de recherche sera construite comme suit :

```
-- PostgresSQL
WHERE user.is_admin::varchar like 'true'
```

Il est possible de saisir n'importe quelle chaîne de caractères dans `$args[0]` et pas seulement des valeurs booléennes.

Attention : Cette recherche n'est pas sensible à la casse. Plusieurs fonctions de formatage sont appelées sur `user.is_admin` avant de tester l'égalité.

2.1.5.3.2.3 Tester si une donnée est présente ou non dans un groupe de données

Exemple : recherche des utilisateurs administrateurs.

Dans cet exemple, l'information se trouve non pas dans la table utilisateur mais dans la table administrateur disposant d'une colonne `user_id` (clé étrangère). Il nous faut utiliser une sous-requête pour récupérer l'ensemble des identifiants de la table administrateur afin de tester si un identifiant utilisateur est effectivement présent dans cette liste.

```
<?php

// soit 'user' une table contenant pas la colonne 'is_admin'
// soit 'admin' une table contenant une colonne 'user_id'

$args = array();
$args[0] = array('', 'true', 'false');
$args[1] = array(_('Tous'),
                 _('Administrateurs'),
                 _('Utilisateurs simples'));

$subquery = 'SELECT user_id FROM admin';

$champs['administrator'] =
    array('colonne' => 'id',
          'table' => 'user',
          'libelle' => _('Administrateur'),
          'type' => 'select',
          'subtype' => 'manualselect',
          'where' => 'insubquery',
          'args' => $args,
          'subquery' => $subquery);

?>
```

Cette configuration permet de créer un champ HTML de type `select` avec trois choix :
– Tous (valeur '') ;
– Administrateurs (valeur `true`) ;
– Utilisateurs simples (valeur `false`).

Le tableau `$args[0]` contient les valeurs associées aux choix. La valeur `true` indique que les identifiants des utilisateurs doivent se trouver dans la sous-requête. La valeur `false` indique qu'ils ne doivent pas se trouver dans la sous-requête. Contrairement à l'exemple « Créer un champ de recherche avec menu déroulant personnalisé », les valeurs ne seront pas recherchées telles quelles dans la base de données et ne doivent surtout pas être modifiées.

En sélectionnant « Administrateurs », la requête SQL de recherche sera construite comme suit :

```
WHERE user.id IN (SELECT user_id FROM admin)
```

2.1.6 Les formulaires

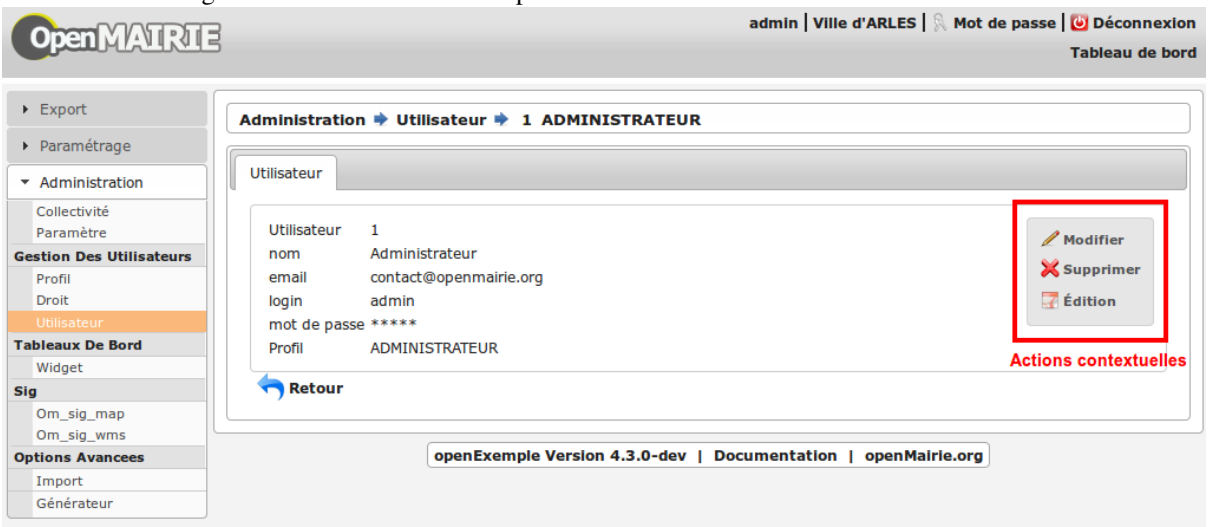
Les formulaires openMairie sont une visualisation d'un objet d'une classe métier.

2.1.6.1 Introduction

Les formulaires permettent la consultation, l'ajout, la modification et la suppression d'enregistrements des tables de la base de données.

Consultation

La consultation d'un élément est construite de la même façon qu'un formulaire. Elle contient une liste d'actions contextuelles configurable. Les données ne sont pas éditables.

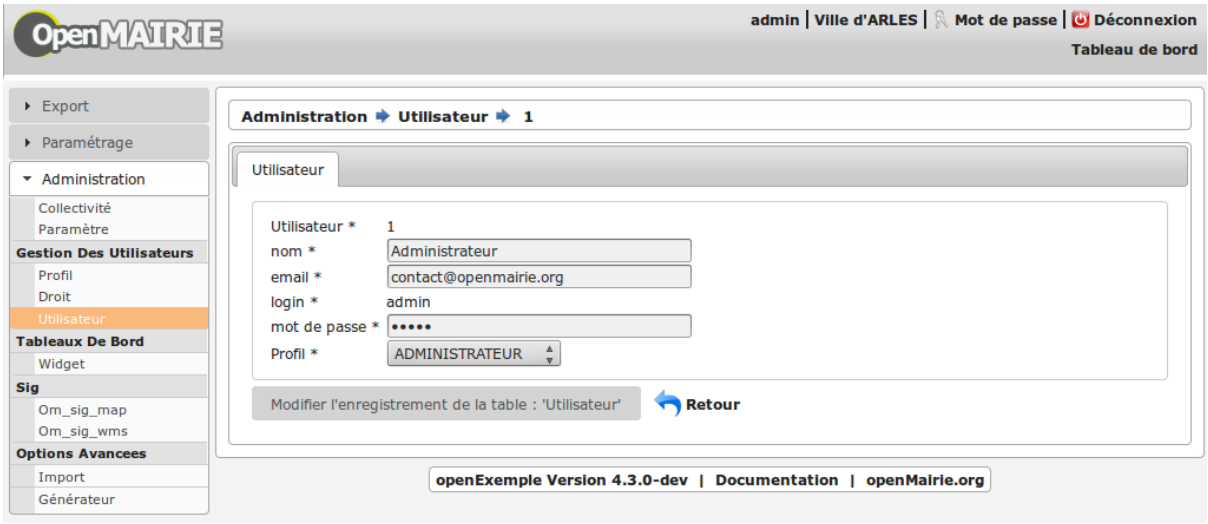


Ajout

L'ajout permet l'éditions de données. Lors de la validation, un traitement spécifique des données est effectué. Si la clé primaire de la table est automatique alors elle est générée.

Modification

L'ouverture d'un élément en modification permet l'éditions de données déjà existantes, lors de la validation du formulaire les données sont traitées, vérifiées puis envoyées dans la base.



Suppression

Accessible depuis la liste des actions contextuelles, une confirmation est demandée pour chaque suppression.

Accès

L'accès aux formulaires se fait depuis un *tableau d'éléments* ou depuis la consultation d'un élément via le menu contextuel.

Par défaut, depuis les tableaux, les actions d'ajout et consultation sont disponible.

2.1.6.2 Description technique

La gestion des formulaires se base sur deux classes :

- formulaire : core/om_formulaire.class.php
- dbform : core/om_dbform.class.php

La classe "formulaire" permet la gestion de l'affichage et "dbform" gère le traitement des données et la liaison à la base de données.

scr/form.php et scr/sousform.php

Ces scripts sont appelés pour afficher un formulaire. Ils instancient l'objet et appellent la méthode formulaire de celui-ci.

Ces scripts prennent plusieurs paramètres :
– obj : nom de la classe pour laquelle on souhaite afficher le formulaire
– action : type d'action (ajout, modification, suppression, consultation)
– idx : identifiant (dans la base de données) de l'élément sur lequel on souhaite effectuer l'action
– retour : deux valeurs possible tab ou form selon l'origine de l'action
Le paramètre "action" peut prendre 4 valeurs :
– 0 : affiche un formulaire d'ajout, le paramètre idx n'est donc pas nécessaire.
– 1 : affiche le formulaire de modification.
– 2 : affiche le formulaire de suppression.
– 3 : affiche le formulaire de consultation.
Les autres paramètres passés permettent de conserver la configuration du tableau d'origine.

2.1.6.3 Description de la classe dbform

class dbform (*$id*, *&$db*, *$DEBUG = false*)

Cette classe est centrale dans l'application. Elle est la classe parente de chaque objet métier. Elle comprend des méthodes de gestion (initialisation, traitement, vérification, trigger) des valeurs du formulaire. Elle fait le lien entre la base de données et le formulaire. Elle contient les actions possibles sur les objets (ajout, modification, suppression, consultation).

Présentation des méthodes de la classe

Les méthodes de dbform peuvent être surchargées dans obj/om_dbform.class.php ainsi que dans toutes les classes métier.

Méthodes d'initialisation de l'affichage du formulaire

dbform.**formulaire** (*$enteteTab*, *$validation*, *$maj*, *&$db*, *$postVar*, *$aff*, *$DEBUG = false*, *$idx*, *$premier = 0*, *$recherche = ""*, *$tricol = ""*, *$idz = ""*, *$selectioncol = ""*, *$advs_id = ""*, *$valide = ""*, *$retour = ""*, *$actions = array()*, *$extra_parameters = array()*)

Méthode d'initialisation de l'affichage de formulaire.

dbform.**sousformulaire** (*$enteteTab*, *$validation*, *$maj*, *&$db*, *$postVar*, *$premiersf*, *$DEBUG*, *$idx*, *$idxformulaire*, *$retourformulaire*, *$typeformulaire*, *$objsf*, *$tricolsf*, *$retour= ""*, *$actions = array()*)

Méthode d'initialisation de l'affichage de sous formulaire.

Ces méthodes instancient un objet "formulaire" et initialisent certains de ses attributs via les méthodes suivantes :

dbform.**setVal** (*&$form*, *$maj*, *$validation*, *&$db*, *$DEBUG = false*)

Permet de définir les valeurs des champs

dbform.**setType** (*&$form*, *$maj*)

Permet de définir le type des champs

dbform.**setLib** (*&$form*, *$maj*)

Permet de définir le libellé des champs

dbform.**setTaille** (*&$form*, *$maj*)

Permet de définir la taille des champs

dbform.**setMax** (*&$form*, *$maj*)

Permet de définir le nombre de caractères maximum des champs

dbform.**setSelect** (*&$form*, *$maj*, *$db*, *$DEBUG = false*)

Méthode qui effectue les requêtes de configuration des champs

dbform.**init_select** (*&$form = null*, *&$db = null*, *$maj*, *$debug*, *$field*, *$sql*, *$sql_by_id*, *$om_validite = false*, *$multiple = false*)

Méthode qui permet la configuration des select et select multiple, elle effectue les requêtes et met en forme le tableau des valeurs à afficher. Il est possible de définir si le champ lié est affecté par une date de validité ou de configurer l'affichage de select_multiple.

dbform.**setOnchange** (*&$form*, *$maj*)

Permet de définir l'attribut "onchange" sur chaque champ

dbform.**setOnkeyup** (*&$form*, *$maj*)

Permet de définir l'attribut "onkeyup" sur chaque champ

dbform.**setOnclick** (*&$form*, *$maj*)

Permet de définir l'attribut "onclick" sur chaque champ

dbform.**setGroupe** (*&$form*, *$maj*)

Permet d'aligner plusieurs champs (obsolète depuis la version 4.3.0)

dbform.**setRegroupe** (*&$form*, *$maj*)

Permet de regrouper les champs dans des fieldset (obsolète depuis la version 4.3.0)

dbform.**setLayout**(*&$form*, *$maj*)

Méthode de mise en page, elle permet de gérer la hiérarchie d'ouverture et fermeture des balises div et fieldset avec les méthodes :

formulaire.**setBloc**(*$champ*, *$contenu*, *$libelle = ''*, *$style = ''*)

permet d'ouvrir/fermer ($contenu=D/F) une balise div sur un champ ($champ), avec un libellé ($libelle) et un attribut class ($style).

–une liste de classes css pour fieldset est disponible : 'group' permet une mise en ligne des champs contenu dans le div et 'col_1 à col_12' permet une mise en page simplifiée (par exemple : "col_1" permet de définir une taille dynamique de 1/12ème de la page , col_6 correspond à 6/12 soit 50% de l'espace disponible).

–il est possible de créer et ajouter des classes css aux différents div afin d'obtenir une mise en page personnalisé.

formulaire.**setFieldset**(*$champ*, *$contenu*, *$libelle = ''*, *$style = ''*)

permet d'ouvrir/fermer ($contenu=D/F) un fieldset sur un champ ($champ), avec une légende ($libelle) et un attribut class ($style).

–une liste de classes css pour fieldset est disponible : 'collapsible' ajoute un bouton sur la légende (jQuery) afin de refermer le fieldset et 'startClosed' idem à la différence que le fieldset est fermé au chargement de la page.

–exemple d'implémentation de la méthode setLayout() sans utiliser les méthodes setGroupe() et setRegroupe() :

```php
<?php
function setLayout(&$form, $maj) {
  //Ouverture d'un div sur une colonne de 1/2 (6/12) de la largeur du
  //conteneur parent
  $form->setBloc('om_collectivite','D',"","col_6");
    //Ouverture d'un fieldset
    $form->setFieldset('om_collectivite','D',_('om_collectivite'),
                       "collapsible");
      //Ouverture d'un div les champs compris entre
      //"om_collectivite" et "actif"
      //la classe group permet d'afficher les champs en ligne
      $form->setBloc('om_collectivite','D',"","group");
      //Fermeture du groupe
      $form->setBloc('actif','F');
    //Fermeture du fieldset
    $form->setFieldset('actif','F','');
  //Fermeture du div de 50%
  $form->setBloc('actif','F');

  //Ouverture d'un div sur une colonne de 1/2 de la largeur du
  //conteneur parent
  $form->setBloc('orientation','D',"","col_6");
    $form->setFieldset('orientation', 'D',
                       _("Parametres generaux du document"),
                       "startClosed");
      $form->setBloc('orientation','D',"","group");
      $form->setBloc('format','F');

      $form->setBloc('footerfont','D',"","group");
      $form->setBloc('footertaille','F');

      $form->setBloc('logo','D',"","group");
      $form->setBloc('logotop','F');
    $form->setFieldset('logotop','F','');
  $form->setBloc('logotop','F');

  //Ouverture d'un div de largeur maximum sur un seul champ
  $form->setBloc('titre','DF',"","col_12");

  //Ouverture d'un div de largeur maximum
```

```
        $form->setBloc('titreleft','D',"","col_12");
          $form->setFieldset('titreleft','D',
                             _("Parametres du titre du document"),
                             "startClosed");
            $form->setBloc('titreleft','D',"","group");
            $form->setBloc('titrehauteur','F');

            $form->setBloc('titrefont','D',"","group");
            $form->setBloc('titrealign','F');
          $form->setFieldset('titrealign','F','');
        $form->setBloc('titrealign','F');

        //Ouverture d'un div de largeur maximum sur un seul champ
        $form->setBloc('corps','DF',"","col_12");

        //Ouverture d'un div de largeur maximum
        $form->setBloc('corpsleft','D',"","col_12");
          $form->setFieldset('corpsleft','D',
                             _("Parametres du corps du document"),
                             "startClosed");
            $form->setBloc('corpsleft','D',"","group");
            $form->setBloc('corpshauteur','F');

            $form->setBloc('corpsfont','D',"","group");
            $form->setBloc('corpsalign','F');
          $form->setFieldset('corpsalign','F','');
        $form->setBloc('corpsalign','F');

        //Ouverture d'un div de largeur maximum sur un seul champ
        $form->setBloc('om_sql','DF',"","col_12");

        //Ouverture d'un div de 1/2 de la largeur du conteneur parent
        $form->setBloc('om_sousetat','D',"","col_6");
          $form->setFieldset('om_sousetat','D',
                             _("Sous etat(s) : selection"),
                             "startClosed");
            $form->setBloc('om_sousetat','D',"","group");
            $form->setBloc('sousetat','F');
          $form->setFieldset('sousetat','F', '');
        $form->setBloc('sousetat','F');

        //Ouverture d'un div de 1/2 de la largeur du conteneur parent
        $form->setBloc('se_font','D',"","col_6");
          $form->setFieldset('se_font','D',
                             _("Sous etat(s) : police / marges / couleur"),
                             "startClosed");
            $form->setBloc('se_font','D',"","group");
            $form->setBloc('se_couleurtexte','F');
          $form->setFieldset('se_couleurtexte','F','');
        $form->setBloc('se_couleurtexte','F');
    }
    ?>
```

Méthodes d'actions

Ces méthodes sont appelées lors de la validation du formulaire.

dbform.**ajouter**(*$val*, *&$db = NULL*, *$DEBUG = false*)
: Cette méthode permet l'insertion de données dans la base, elle appelle toutes les méthodes de traitement, vérification et méthodes spécifiques à l'ajout.

dbform.**modifier**(*$val = array()*, *&$db = NULL*, *$DEBUG = false*)
: Cette méthode permet la modification de données dans la base, elle appelle toutes les méthodes

de traitement et vérification des données retournées par le formulaire.

dbform.**supprimer**(*$val = array()*, *&$db = NULL*, *$DEBUG = false*)
Cette méthode permet la suppression de données dans la base, elle appelle toutes les méthodes de traitement et vérification des données retournées par le formulaire.

Gestion des transactions lors de l'appel aux méthodes d'actions

Afin de verifier les erreurs de base de données, la méthode isError est appelée, si la valeur true lui est passée en second paramètre elle ne stop pas l'execution mais retour true ou false. Cela dans le but d'appeler ces méthodes sur des objets métier instanciés manuellement dans des contextes qui n'utilise pas la classe formulaire. Exemple : lors de la création d'un web service qui instancierait une classe, si une erreur de base de données se produit, le script s'arrête et aucun message ne peut être transmis au client du web service, ce qui ne se produit pas si le second paramètre est défini à true.

Il est important d'instancier un objet métier et d'appeler les méthodes ajouter, modifier ou supprimer pour effectuer un changement sur celui-ci car toutes les méthodes de trigger seront appelées.

Méthodes appelées lors de la validation

dbform.**setValFAjout**(*$val = array()*)
Méthode de traitement des données retournées par le formulaire (utilisé lors de l'ajout)

dbform.**setvalF**(*$val = array()*)
Méthode de traitement des données retournées par le formulaire

dbform.**verifier**(*$val = array()*, *&$db = NULL*, *$DEBUG = false*)
Méthode de vérification des données et de retour d'erreurs

dbform.**verifierAjout**(*$val = array()*, *&$db = NULL*)
Méthode de vérification des données et de retour d'erreurs (utilisé lors de l'ajout)

dbform.**setId**(*&$db = NULL*)
Initialisation de la clé primaire (si clé automatique lors de l'ajout)

dbform.**cleSecondaire**(*$id*, *&$db = NULL*, *$val = array()*, *$DEBUG = false*)
Cette méthode est appelée lors de la suppression d'un objet, elle permet de vérifier si l'objet supprimé n'est pas lié à une autre table pour en empêcher la suppression.

dbform.**triggerajouter**(*$id*, *&$db = NULL*, *$val = array()*, *$DEBUG = false*)
Permet d'effectuer des actions avant l'insertion des données dans la base

dbform.**triggerajouterapres**(*$id*, *&$db = NULL*, *$val = array()*, *$DEBUG = false*)
Permet d'effectuer des actions après l'insertion des données dans la base

dbform.**triggermodifier**(*$id*, *&$db = NULL*, *$val = array()*, *$DEBUG = false*)
Permet d'effectuer des actions avant la modification des données dans la base

dbform.**triggermodifierapres**(*$id*, *&$db = NULL*, *$val = array()*, *$DEBUG = false*)
Permet d'effectuer des actions après la modification des données dans la base

dbform.**triggersupprimer**(*$id*, *&$db = NULL*, *$val = array()*, *$DEBUG = false*)
Permet d'effectuer des actions avant la modification des données dans la base

dbform.**triggersupprimerapres**(*$id*, *&$db = NULL*, *$val = array()*, *$DEBUG = false*)
Permet d'effectuer des actions après la modification des données dans la base

2.1.6.4 Description de la classe formulaire

class formulaire(*$unused = NULL*, *$validation*, *$maj*, *$champs = array()*, *$val = array()*, *$max = array()*)
Cette classe permet une gestion complète de l'affichage d'un formulaire.

Les méthodes de core/om_formulaire.class.php peuvent être surchargées dans obj/om_formulaire.class.php

Méthodes d'affichage de widgets

Les widgets sont des éléments de formulaire, ils sont composés d'un ou plusieurs champs. Chaque méthode permet d'afficher un seul widget.

`formulaire.text()`
: champ texte (format standard)

`formulaire.hidden()`
: champ non visible avec valeur conservée

`formulaire.password()`
: champ password

`formulaire.textdisabled()`
: champ texte non modifiable (grisé)

`formulaire.textreadonly()`
: champ texte non modifiable

`formulaire.hiddenstatic()`
: champ non modifiable, la valeur est récupéré par le formulaire.

`formulaire.hiddenstaticnum()`
: champ numérique non modifiable et valeur récupérer

`formulaire.statiq()`
: Valeur affichée et non modifiable

`formulaire.affichepdf()`
: récupère un nom d'objet (un scan pdf)

`formulaire.checkbox()`
: case à cocher valeurs possibles : `True` ou `False`

`formulaire.checkboxstatic()`
: affiche Oui/Non, non modifiable (mode consultation)

`formulaire.checkboxnum()`
: cochée = 1 , non cochée = 0

`formulaire.http()`
: lien http avec target = _blank (affichage dans une autre fenêtre)

`formulaire.httpclick()`
: lien avec affichage dans la même fenêtre.

`formulaire.date()`
: date modifiable avec affichage de calendrier jquery

`formulaire.date2()`
: date modifiable avec affichage de calendrier jquery pour les sous-formulaires

`formulaire.hiddenstaticdate()`
: date non modifiable Valeur récupéré par le formulaire

`formulaire.datestatic()`
: affiche la date formatée, non modifiable (mode consultation)

`formulaire.textarea()`
: affichage d un textarea

`formulaire.textareamulti()`
: textarea qui récupère plusieurs valeurs d'un select

`formulaire.textareahiddenstatic()`
: affichage non modifiable d'un textarea et récupération de la valeur

`formulaire.pagehtml()`
: affichage d'un textarea et transforme les retours charriot en </ br>

`formulaire.select()`
: champ select

`formulaire.selectdisabled()`
: champ select non modifiable

`formulaire.selectstatic()`
: affiche la valeur de la table liée, non modifiable (mode consultation)

formulaire.**selecthiddenstatic**()
: affiche la valeur de la table liée, non modifiable ainsi que la valeur dans un champ hidden

formulaire.**select_multiple**()
: affiche un select multiple, les valeurs passées au formulaires doivent être séparées par une virgule.

formulaire.**select_multiple_static**()
: affiche seulement les valeurs d'un select multiple, les valeurs passées au formulaires doivent être séparées par une virgule.

formulaire.**comboG**()
: permet d'effectuer une corrélation entre un groupe de champ et un identifiant dans les formulaires

formulaire.**comboG2**()
: permet d'effectuer une corrélation entre un groupe de champ et un identifiant dans les sous formulaires

formulaire.**comboD**()
: permet d'effectuer une corrélation entre un groupe de champ et un identifiant dans les formulaires

formulaire.**comboD2**()
: permet d'effectuer une corrélation entre un groupe de champ et un identifiant dans les sous formulaires

formulaire.**upload**()
: fait appel à spg/upload.php pour télécharger un fichier

formulaire.**upload2**()
: fait appel à spg/upload.php pour télécharger un fichier dans un sous formulaire

formulaire.**voir**()
: fait appel à spg/voir.php pour visualiser un fichier

formulaire.**voir2**()
: fait appel à spg/voir.php pour visualiser un fichier depuis un sous formulaire

formulaire.**localisation**()
: fait appel à spg/localisation.php

formulaire.**localisation2**()
: fait appel à spg/localisation.php

formulaire.**rvb**()
: fait appel à spg/rvb.php pour affichage de la palette couleur

formulaire.**rvb2**()
: fait appel à spg/rvb.php pour affichage de la palette couleur

formulaire.**geom**()
: ouvre une fenêtre tab_sig.php pour visualiser ou saisir une géométrie (selon l'action) la carte est définie en setSelect

Les widgets comboG, comboD, date, upload, voir et localisation sont à mettre dans les formulaires. Les contrôle comboG2, comboD2, date2, upload2, voir2 et localisation sont à mettre dans les sous formulaires.

Les widgets font appel des scripts d'aide à la saisie stockés dans le répertoire /spg, ils sont appelés par js/script.js. Ce script peut être surchargé dans app/js/script.js.

spg/combo.php

Ce programme est appelé par le champ comboD, comboG, comboD2, comboG2, le paramétrage se fait dans les fichiers :
– dyn/comboparametre.inc.php
– dyn/comboretour.inc.php
– dyn/comboaffichage.inc.php
spg/localisation.php et js/localisation.js

ce programme est liée au champ formulaire "localisation".

spg/voir.php

Ce script est associé au champ "upload".

Ce sous programme permet de visualiser un fichier téléchargé sur le serveur (pdf ou image).

spg/upload.php

Ce script utilise la classe core/upload.class.php (composant openMairie).

Le paramétrage des extensions téléchargeables se fait dans dyn/config.inc.php. Le paramétrage de la taille maximale des fichiers téléchargeables se fait dans la classe métier de l'objet.

spg/rvb.php et js/rvb.js

Ce script est associé au champ "rvb" et affiche une palette de couleur pour récupérer un code rvb.

Les méthodes de construction et d'affichage

Le formulaire est constitué de div, fieldset et de champs les méthodes suivantes permettent une mise en page structurée.

`formulaire.`**`entete`**`()`
: ouverture du conteneur du formulaire.

`formulaire.`**`enpied`**`()`
: fermeture du conteneur du formulaire.

`formulaire.`**`afficher`**`()`
: affichage des champs, appelle les méthodes suivante :

`formulaire.`**`debutFieldset`**`()`
: ouverture de fieldset.

`formulaire.`**`finFieldset`**`()`
: fermeture de fieldset

`formulaire.`**`debutBloc`**`()`
: ouverture de div.

`formulaire.`**`finBloc`**`()`
: fermeture de div.

`formulaire.`**`afficherChamp`**`()`
: affichage de champ.

Les méthodes assesseurs changent les valeurs des attributs de l'objet formulaire

Ces méthodes sont appelées depuis les classes métier, elles permettent la configuration du formulaire.

`formulaire.`**`setType`**`()`
: type de champ

`formulaire.`**`setVal`**`()`
: valeur du champ

`formulaire.`**`setLib`**`()`
: libellé du champ

`formulaire.`**`setSelect`**`()`
: permet de remplir les champs select avec la table liée

`formulaire.`**`setTaille`**`()`
: taille du champ

`formulaire.`**`setMax`**`()`
: nombre de caractères maximum acceptés

`formulaire.`**`setOnchange`**`()`
: permet de définir des actions sur l'événement "onchange"

`formulaire.`**`setKeyup`**`()`
: permet de définir des actions sur l'événement "onkeyup"

`formulaire.`**`setOnclick`**`()`
: permet de définir des actions sur l'événement "onclick"

`formulaire.`**`setvalF`**`()`
permet de traiter les données avant insert/update dans la base de données

`formulaire.`**`setGroupe`**`()`
(obsolète depuis 4.3.0)

`formulaire.`**`setRegroupe`**`()`
(obsolète depuis 4.3.0)

`formulaire.`**`setBloc`**`(`*$champ*, *$contenu*, *$libelle* = '', *$style* = '')
permet d'ouvrir/fermer ($contenu=D/F/DF) une balise div sur un champ ($champ), avec un libellé ($libelle) et un attribut class ($style).

`formulaire.`**`setFieldset`**`(`*$champ*, *$contenu*, *$libelle* = '', *$style* = '')
permet d'ouvrir/fermer ($contenu=D/F/DF) un fieldset sur un champ ($champ), avec une légende ($libelle) et un attribut class ($style).

2.1.7 Les actions vers formulaires

Les liens vers les formulaires sont principalement dans les tableaux et formulaires de consultation d'objets.

2.1.7.1 Actions des tableaux

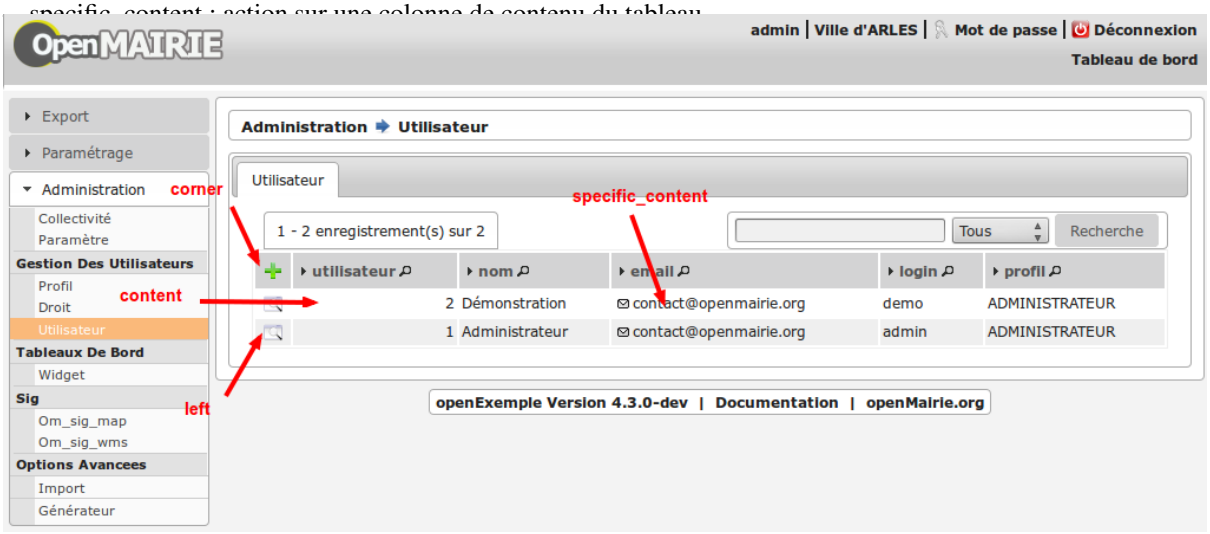
La surcharge des actions de tableaux se fait via les scripts `sql/sgbd/objet.inc.php`.

L'ajout d'actions se présente de cette façon :

```
<?php
    // Actions en coin ('corner') : ajouter
    $tab_actions['corner']['ajouter'] =
        array('lien' => 'form.php?obj='.$obj.'&amp;action=0',
              'id' => '&amp;advs_id='.$advs_id.'&amp;tricol='.$tricol.
                      '&amp;valide='.$valide.'&amp;retour=tab',
              'lib' => '<span class="om-icon om-icon-16 om-icon-fix add-16"
                          title="'._('Ajouter').'">'._('Ajouter').'</span>',
              'rights' => array('list' => array($obj, $obj.'_ajouter'),
                                'operator' => 'OR'),
              'ordre' => 10,);

    // Actions à gauche ('left'): consulter
    $tab_actions['left']['consulter'] =
        array('lien' => 'form.php?obj='.$obj.'&amp;action=3'.'&amp;idx=',
              'id' => '&amp;premier='.$premier.'&amp;advs_id='.$advs_id.
                      '&amp;recherche='.$recherche1.'&amp;tricol='.$tricol.
                      '&amp;selectioncol='.$selectioncol.'&amp;valide='.
                      $valide.'&amp;retour=tab',
              'lib' => '<span class="om-icon om-icon-16 om-icon-fix
                        consult-16" title="'._('Consulter').'">'.
                        _('Consulter').'</span>',
              'rights' => array('list' => array($obj, $obj.'_consulter'),
                          'operator' => 'OR'),
              'ordre' => 10,);

    // Action sur la cinquième colonne de contenu
    $tab_actions['specific_content'][4] =
        array('lien' => 'form.php?obj='.$obj.'&amp;action=2'.'&amp;idx=',
              'id' => '&amp;premier='.$premier.'&amp;advs_id='.$advs_id.
                      '&amp;recherche='.$recherche1.'&amp;tricol='.$tricol.
                      '&amp;selectioncol='.$selectioncol.
                      '&amp;valide='.$valide.'&amp;retour=tab',
              'lib' => '<span class="om-icon om-icon-16 om-icon-fix
                        delete-16" title="'._('Consulter').'">'.
                        _('Consulter').'</span>',
```

```
                'rights' => array('list' => array($obj, $obj.'_consulter'),
                                  'operator' => 'OR'),
                'ordre' => 10,);

    ?>
```

Plusieurs emplacements d'actions existent :
– corner : actions dans la première cellule du tableau
– left : action situées dans la première colonne, disponibles pour chaque élément du tableau
– content : action sur le contenu du tableau
– specific_content : action sur une colonne de contenu du tableau



Les actions par défaut

Par défaut seules les actions `ajouter` et `consulter` sont disponibles depuis les tableaux.

Créer de nouvelles actions

La création d'actions pour un tableau particulier se fait depuis le répertoire `sql/sgbd/`.

Les actions doivent se définir dans les fichier `objet.inc.php` de la manière suivante :

```
<?php

$tab_actions['left']['modifier'] =
    array('lien' => 'form.php?obj='.$obj.'&amp;action=1'.'&amp;idx=',
          'id' => '&amp;premier='.$premier.'&amp;advs_id='.$advs_id.'&amp;recherche='.$recherche1
          'lib' => '<span class="om-icon om-icon-16 om-icon-fix edit-16" title="'._('Modifier').'
          'rights' => array('list' => array($obj, $obj.'_modifier'), 'operator' => 'OR'),
          'ordre' => 20,);

?>
```

2.1.7.1.2.1 Définition de l'action

La première clé de `$tab_actions` permet choisir la position d'affichage :
– `corner` pour les actions en coin ;
– `left` pour les actions de gauche.

Note : Depuis la version 4.3.0 d'openMairie, il est désormais possible d'afficher plusieurs actions dans le coin du tableau (au niveau de l'action `ajouter`).

La seconde clé de `$tab_actions` permet de définir la nouvelle action. Cette clé doit être différente de : `ajouter`, `consulter`, `modifier` et `supprimer`.

Les clés `lien`, `id` et `lib` s'utilise de la même manière qu'avant.

2.1.7.1.2.2 Définition du mode d'affichage en sous-tableau La clé `ajax` permet d'indiquer si l'action doit être affichée en ajax ou non dans les sous-tableaux :
– `true`, l'action utilisera la fonction `ajaxIt()` ;
– `false`, l'action n'utilisera pas la fonction `ajaxIt()`.

2.1.7.1.2.3 Définition de l'ordre d'affichage La clé `ordre` permet de déterminer l'ordre d'affichage par rapport aux autres actions.

Chaque action dispose d'une valeur numérique permettant de définir sa place au sein d'une position. L'action numéro 1 s'affichera en premier, l'action numéro 10 s'affichera après les actions de numéro inférieur, etc.

Ordre des actions par défaut d'openMairie :
– ajouter à pour ordre 10 dans la position `corner` ;
– consulter à pour ordre 10 dans la position `left`.
Si la position `corner` est sélectionnée :
– 9, l'action s'affichera avant l'action `ajouter` ;
– 11, l'action s'affichera après l'action `ajouter`.
Si la position `left` est sélectionnée :
– 9, l'action s'affichera avant l'action `consulter` ;
– 11, l'action s'affichera après l'action `consulter`.

2.1.7.1.2.4 Définition des droits d'affichage La clé `rights` permet de définir le ou les droits nécessaires à l'utilisateur pour visualiser cette action. Cette clé est optionnelle. Si `rights` n'existe pas, tous les utilisateurs pourront visualiser cette action s'ils peuvent visualiser le tableau correspondant.

La clé `list` permet de définir le tableau des droits nécessaire.

La clé `operator` permet de définir l'opérateur utilisé pour pour vérifier les droits de la liste `list` :
– `OR`, l'utilisateur doit avoir au moins un droit ;
– `AND`, l'utilisateur doit avoir tous les droits.

2.1.7.2 Actions du menu contextuel de la consultation

La configuration des actions du menu contextuel des formulaires en consultation se fait via les scripts `sql/sgbd/objet.form.inc.php`

Dans ces scripts, peuvent être surchargés, la liste des champs (ordre ou champs affichés), requêtes sql permettant de remplir les widget de formulaires ainsi que les actions du menu contextuel.

L'ajout d'une action se présente de cette façon :

```
<?php
$portlet_actions['edition'] = array(
    'lien' => '../pdf/pdflettretype.php?obj=om_utilisateur&amp;idx=',
    'id' => '',
    'lib' => '<span class="om-prev-icon om-icon-16 om-icon-fix pdf-16"
                    title="'._('Edition').'">'._('Edition').'</span>',
    'ajax' => false,
    'ordre' => 21,
);
?>
```

2.1.8 La méthode

Le développement consiste à créer des objets métier (/obj) qui surchargent la classe abstraite om_dbformdyn.class.php et à modifier les valeurs par défaut des variables dans les fichiers sql (nom_objet.inc et nom_objet.form.inc)

Voir aussi le *générateur* pour automatiser les scripts métier.

Il est décrit ensuite les 2 objets : objet de connexion db et l'objet formulaire form.

2.1.8.1 Surcharger les classes openMairie

Il vaut mieux utiliser le générateur pour initialiser les classes métiers.

Le générateur surcharge la classe om_dbformdyn.class.php par rapport aux informations de la base

```
classe abstraite <- classe métier générée <- classe métier 1 <- classe métier 2 ...
openMairie             depuis la base


om_dbformdyn.class.php <- gen/obj/nom_objet.class.php <- obj/nom_objet.class.php
```

Exemple avec concession d'openCimetiere

```
om_dbformdyn.class.php
        <- gen/obj/emplacement.class.php
            <-/obj/emplacement.class.php <- /obj/concession.class.php
```

La classe dbformdyn.class.php fait appel à la classe formulaire.dyn.class.php pour afficher le formulaire.

Il est créé 2 objets :
– un objet db qui fait la connexion avec la base
– un objet form qui décrit le formulaire

2.1.8.2 L'objet db

db est l'objet de connexion a la base dont les propriétés sont les suivantes

```
DB_pgsql Object

(
[phptype] => pgsql
    [dbsyntax] => pgsql
    [features] => Array (
                    [limit] => alter
                    [new_link] => 4.3.0
                    [numrows] => 1
                    [pconnect] => 1
                    [prepare] =>
                    [ssl] => 1
                    [transactions] => 1 )
                    [errorcode_map] => Array ( )
                    [connection] => Resource id #19
                    [dsn] => Array (
                            [phptype] => pgsql
                            [dbsyntax] => pgsql
                            [username] => postgres
                            [password] => postgres
                            [protocol] => tcp
                            [hostspec] => localhost
                            [port] => 5432
                            [socket] =>
```

```
                        [database] => sig
                        [title] => Openmairie Exemple PostGreSQL schema SIG
                        [formatdate] => AAAA-MM-JJ
                        [schema] => openmairie
                    )
                    [autocommit] => 1
                    [transaction_opcount] => 0
                    [affected] => 0
                    [row] => Array ([20] => 10 )
                    [_num_rows] => Array ( [20] => 10 )
                    [fetchmode] => 1
                    [fetchmode_object_class] => stdClass
                    [was_connected] =>
                    [last_query] => select * from openmairie.om_parametre where om_collectivite=2
                    [options] => Array (
            [result_buffering] => 500
                        [persistent] =>
                        [ssl] =>
            [debug] => 2
            [seqname_format] => %s_seq
            [autofree] =>
            [portability] => 63
            [optimize] => performance
            )
                    [last_parameters] => Array ( )
                    [prepare_tokens] => Array ( )
                    [prepare_types] => Array ( )
                    [prepared_queries] => Array ( )
                    [_last_query_manip] =>
                    [_next_query_manip] =>
                    [_debug] =>
                    [_default_error_mode] =>
                    [_default_error_options] =>
                    [_default_error_handler] =>
                    [_error_class] => DB_Error
                    [_expected_errors] => Array ( )
)
```

2.1.8.3 L'objet form

form est l'objet formulaire dont les propriétés sont les suivantes

```
formulaire Object (
    [enteteTab] =>
    [val] => Array (
            [om_parametre] => 1
            [libelle] => maire
            [valeur] => O PENMAIRIE
            [om_collectivite] => 1 )
    [type] => Array (
            [om_parametre] => text
            [libelle] => text
            [valeur] => text
            [om_collectivite] => text )
    [taille] => Array (
            [om_parametre] => 11
            [libelle] => 20
            [valeur] => 50
            [om_collectivite] => 11 )
    [max] => Array (
            [om_parametre] => 11
```

```
            [libelle] => 20
            [valeur] => 50
            [om_collectivite] => 11 )
    [lib] => Array (
            [om_parametre] => Om_parametre
            [libelle] => Libelle
            [valeur] => Valeur
            [om_collectivite] => Om_collectivite )
    [groupe] => Array (
            [om_parametre] =>
            [libelle] =>
            [valeur] =>
            [om_collectivite] => )
    [select] => Array (
            [om_parametre] =>  Array ([0] => [1] => )
            [libelle] => Array ( [0] => [1] => )
            [valeur] => Array ( [0] => [1] => )
            [om_collectivite] => Array ( [0] => [1] => ) )
    [onchange] => Array (
            [om_parametre] =>
            [libelle] =>
            [valeur] =>
            [om_collectivite] => )
    [onkeyup] => Array (
            [om_parametre] =>
            [libelle] =>
            [valeur] =>
            [om_collectivite] => )
    [onclick] => Array (
            [om_parametre] =>
            [libelle] =>
            [valeur] =>
            [om_collectivite] => )
    [regroupe] =>
    [correct] =>
)
```

2.1.9 Les éditions

openMairie permet d'effectuer des éditions au format PDF. Ces éditions sont paramétrées depuis l'interface du logiciel. Pour chaque édition PDF des champs de fusion permettent de récupérer dynamiquement les données à imprimer.

Les éditions sont accessibles dans le menu par

```
- parametrage -> etat
- parametrage -> sousetat
- parametrage -> lettretype
```

Depuis la version 4 d'openMairie, les éditions sont conservées dans 3 tables

```
- om_etat : pour les états
- om_sousetat : pour les sous-états
- om_lettretype : pour les lettres types
```

Contrairement aux états et aux lettres-types, les tableaux de bord PDF sont stockés dans un fichier : nom_objet.pdf.inc .

2.1.9.1 Actif, non actif

Les sous-etats sont liés a un ou plusieurs état.

Les états, sous-états, et lettre type peuvent être “actif” ou “non-actif”.

Par défaut sont pris en compte :

1 - l’édition “actif” de la collectivité

2 - l’édition “actif” de la multicollectivité

3 - l’édition “non-actif” de la multicollectivité

Les éditions d’une collectivité ayant le statut “non-actif” ne sont pas prises en compte.

2.1.9.2 Paramétrer des états

Il est conseillé d’utiliser l’assistant état du générateur.

Les paramètres sont les suivants

```
orientation portrait ou paysage
format="A4", A3
position et nom  du logo
titre de l état
position et caractéristiques du titre
corps de l état
position et caractéristiques du corps
la requête SQL
les sous-états associés et les caractéristiques
```

Pour le corps et le titre, les zones entre crochets (exemple [nom]) sont les champs de fusion, sélectionnés par la requête SQL.

Ces champs de fusion peuvent être mis en majucule ou en minuscule selon les besoins grâce aux balises <MAJ></MAJ> et <min></min>.

EX. :

Requête SQL = SELECT nom as nom FROM A ;

Cette phrase dans un PDF ‘Lorem [nom] dolor sit amet, <MAJ>[nom]</MAJ> adipiscing elit. Curabitur feugiat.’ deviendra ‘Lorem nom dolor sit amet, NOM adipiscing elit. Curabitur feugiat.’

Les variables commençant par “&” sont celles définies dans dyn/varpdf.inc (exemple &aujourdhui) et dans la table om_parametre.

2.1.9.3 Paramétrer des sous-états

Il est conseillé d’utiliser l’assistant sous-etat du générateur.

Les paramètres sont les suivants

```
texte et caractéristique du Titre
Intervalle avant et après le tableau
Entête de tableau (nom de colonne)
caractéristique du tableau
caractéristique des cellules
total, moyenne, nombre
requête SQL
```

Pour le titre, les zones entre crochets sont les champs de fusion, sélectionnés par la requête.

Les variables commençant par “&” sont celles définies dans dyn/varpdf.inc (exemple &aujourdhui) et dans la table om_parametre.

2.1.9.4 Paramétrer des lettres-type

Il est conseillé d'utiliser l'assistant lettre-type du générateur.

Les paramètres sont les suivants

```
orientation portrait ou paysage
format="A4", A3
position et nom  du logo
titre de la lettre
position et caractéristiques du titre
corps de la lettre
position et caractéristiques du corps
la requête SQL
```

Pour le corps et le titre, les zones entre crochets sont les champs de fusion, sélectionnés par la requête.

Les variables commençant par "&" sont celles définies dans dyn/varlettretypepdf.inc (exemple &aujourdhui) et dans la table om_parametre.

2.1.9.5 Paramétrer des édition PDF

Un état PDF peut être généré par le générateur (option).

L'édition est paramétrée dans un fichier sql/sgbd/nom_objet.pdf.inc et dans la

Les paramètres sont les suivants

```
texte et caractéristique du Titre
Entête de tableau (nom de colonne)
caractéristique du tableau
caractéristique des cellules
total, moyenne, nombre
requête SQL
```

Pour le titre, les zones entre crochets sont les champs de fusion, sélectionnés par la requête.

Les variables commençant par "&" sont celles définies dans dyn/varpdf.inc (exemple &aujourdhui) et dans la table om_parametre.

2.1.9.6 Paramétrer les étiquettes

Les zones entre crochets sont les champs de fusion sélectionnés par la requête. Les variables (exemple &aujourdhui) sont celles définies dans dyn/varetiquettepdf.inc et dans la table om_parametre.

Il y aura une integration depuis l'utilisation d'openPersonnalite dans une prochaine version openMairie.

2.1.9.7 L'éditeur WYSIWYG

Un éditeur avancé a été mis en place dans cette version openMairie afin de permettre à l'utilisateur de définir des mises en forme complexes. (tinymce)

2.1.9.8 Les scripts PDF

Les scripts sont dans le répertoire **pdf/** et sont appelés par le framework sous la forme

```
pdfetat.php?obj=nom_etat&idx=enregistrement_a_editer
```

les scripts sont les suivants

```
pdfetat.php : état et sous-état
pdf.php : édition PDF
pdfetiquette.php : étiquette
pdflettretype.php
```

pdfEtiquette sera repris dans une prochaine version d'openMairie

specifique openCourrier pour ecriture sur pdf

```
fpdf_tpl.php
fpdi.php
fpdi2tcpdf_bridge.php
fpdi_pdf_parser.php
histo.htm
pdf_context.php
pdf_parser.php
testfpdi.php
```

Il n'est pas prévu d'intégration dans la prochaine version.

2.1.9.9 Composants

/core

Les scripts ci dessous sont les classes qui interfacent openmairie avec fpdf

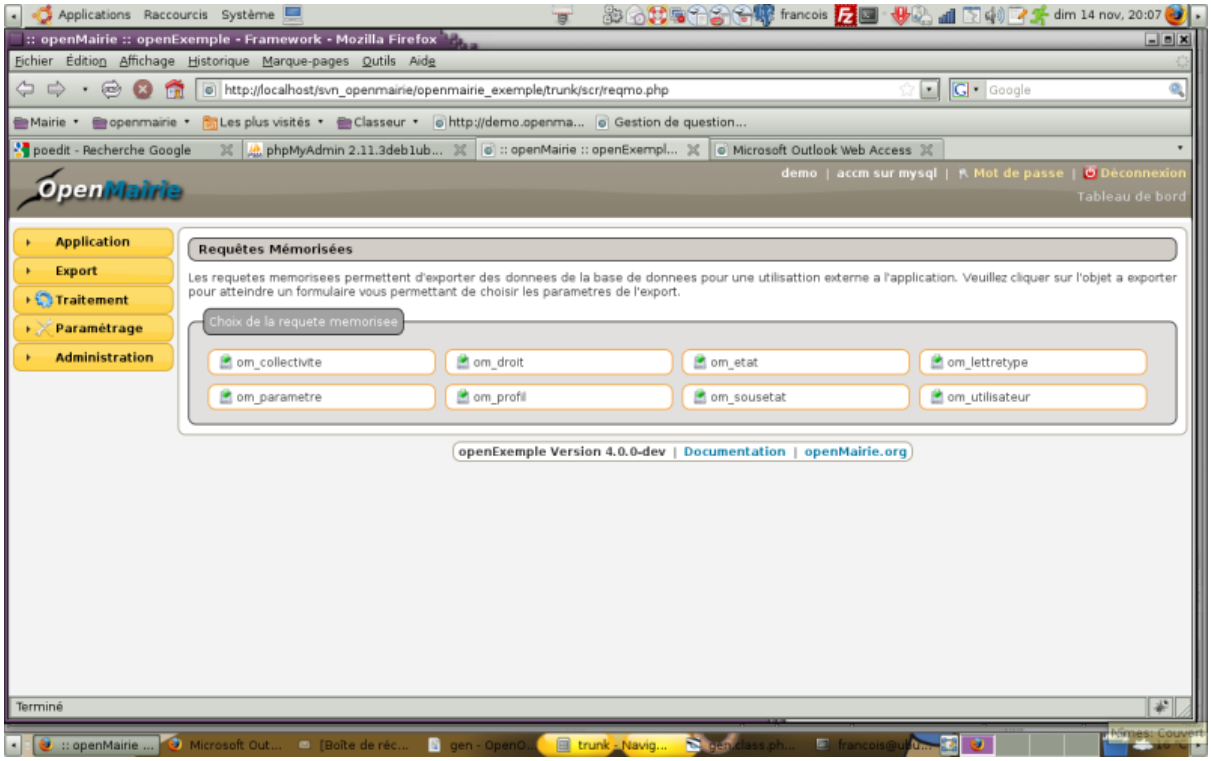
```
fpdf_etat.php
fpdf_etiquette.php
db_fpdf.php
```

php/fpdf

A ce niveau se situe le composant fpdf

2.1.10 Les requêtes memorisées

les requêtes mémorisées permettent au développeur de fournir un ensemble de requêtes :
– mémorisées
– accessible dans le menu export -> requêtes
– paramétrables par l'utilisateur
– permettant un affichage html en tableau ou un transfert au format csv sur tableur (choix du séparateur à l utilisateur)
menu export-> requete


Applications Raccourcis Système
francois
dim 14 nov, 20:07
:: openMairie :: openExemple - Framework - Mozilla Firefox
Fichier Édition Affichage Historique Marque-pages Outils Aide
http://localhost/svn_openmairie/openmairie_exemple/trunk/scr/reqmo.php
Mairie openmairie Les plus visités Classeur http://demo.openma... Gestion de question...
poedit - Recherche Google
phpMyAdmin 2.11.3deb1ub...
:: openMairie :: openExempl...
Microsoft Outlook Web Access
OpenMairie
demo | accm sur mysql | Mot de passe | Déconnexion
Tableau de bord
Application
Export
Traitement
Paramétrage
Administration
Requêtes Mémorisées
Les requetes memorisees permettent d'exporter des donnees de la base de donnees pour une utilisattion externe a l'application. Veuillez cliquer sur l'objet a exporter pour atteindre un formulaire vous permettant de choisir les parametres de l'export.
Choix de la requete memorisee
om_collectivite
om_droit
om_etat
om_lettretype
om_parametre
om_profil
om_sousetat
om_utilisateur
openExemple Version 4.0.0-dev | Documentation | openMairie.org
Terminé

2.1.10.1 Description du parametrage

Les parametres de reqmo sont :

$reqmo['libelle'] contient le libéllé affiché en haut
$reqmo['sql'] contient la requete SQL.
Dans la requete, les paramétres sont mis entre []
et ils sont définis en dessous sous la forme reqmo[parametre]=.

"checked" : la colonne est affiché ou non
"un tableau" array(a,b) et le choix a ou b est donné à l utilisateur de requete
"une requete sql" : le choix se fait dans la table du select

La requete executée est celle qui est reconstituée avec les zones sasisies par l'utilisateur

Enfin, l'utilisateur choisit soit un affichage soit en tableau, soit en csv avec un choix de séparateur.

Il n y a pas d'outil de fabrication de requête à part l'option du générateur (voir chapitre sur le *générateur*)

2.1.10.2 Exemple

voies sous openCimetiere

```
$reqmo['libelle']=" Voies par cimetiere ";

$reqmo['sql']=" select voie,voietype,voielib,
                [zonetype],[zonelib],
                [cimetierelib]
                from voie
                inner join zone
                on voie.zone=zone.zone
                inner join cimetiere
                on zone.cimetiere=cimetiere.cimetiere
                where cimetiere.cimetiere = [cimetiere] order by [tri]";

$reqmo['tri']= array('voielib',
```

```
                'zonelib'
                );
$reqmo['zonetype']="checked";
$reqmo['zonelib']="checked";
$reqmo['cimetierelib']="checked";
$reqmo['cimetiere']="select cimetiere,concat(cimetiere,' ',
        cimetierelib) from cimetiere";
```

2.1.11 La gestion des accès

Le framework fournit un gestionnaire d'accès configurable dans les menus :
– administration -> profil
– administration -> droit
– administration -> utilisateur
Les accès sont conservés dans les tables du même nom.

2.1.11.1 Les tables

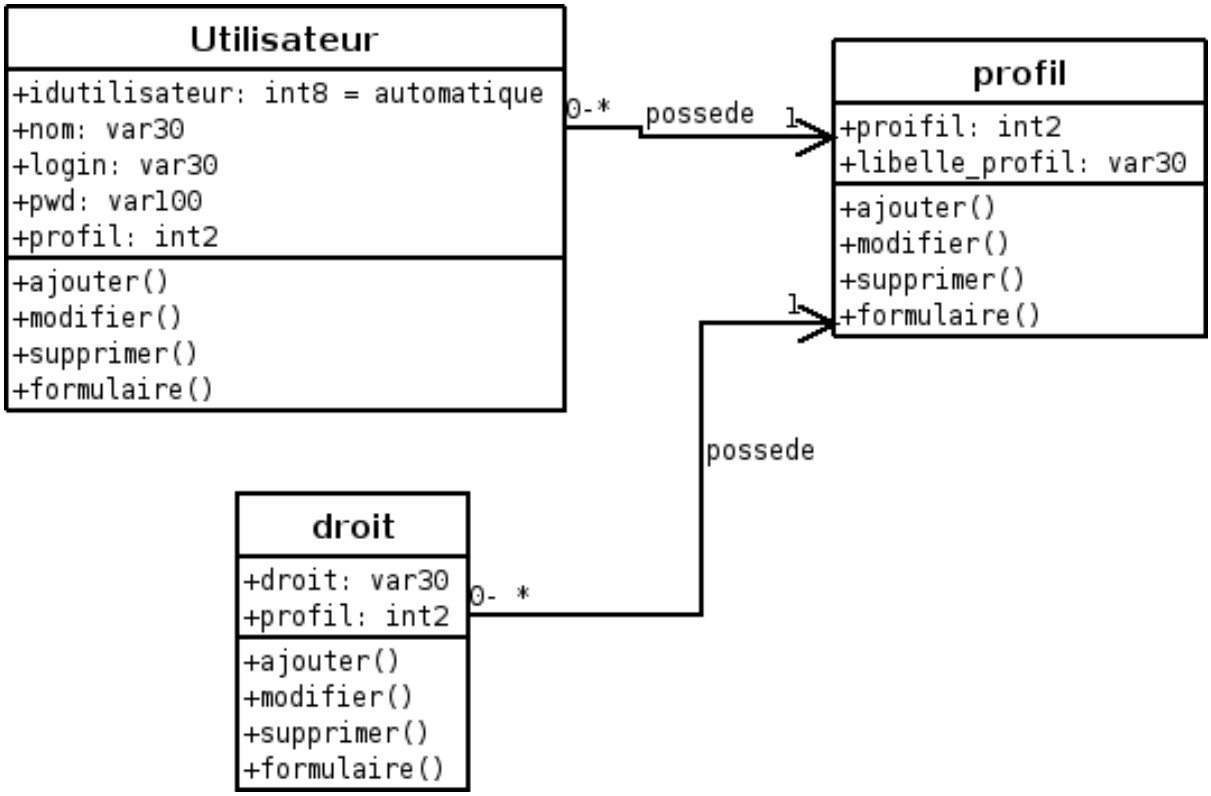
La gestion des accès est gérée avec 3 tables :

om_profil : gestion par défaut de 5 profils :
– administrateur (5)
– super utilisateur (4)
– utilisateur (3)
– utilisateur limité (2)
– consultation (1)
Les profils sont hiérarchiques, le profil 5 étant le plus élevé, il a accès à toutes les actions des profils inférieurs.

om_droit : à chaque profil est affecté un ou plusieurs droits.

om_utilisateur : cette table permet de donner un login, un mot de passe et un profil à chaque utilisateur.

Diagramme de classe



2.1.11.2 Les règles

Le droit sur un objet porte le nom de l'objet, pour chaque objet il existe deux types de droits :
– généraux : il n'est composé que du nom de l'objet et permet d'accéder à toutes les actions sur celui-ci.
– spécifique : il se compose du nom de l'objet puis d'un suffixe.
Détails des suffixes de droits :
– _tab : permet d'accéder au tableau
– _ajouter : permet d'ajouter un objet
– _modifier : permet de modifier l'objet
– _supprimer : permet de supprimer l'objet
– _consulter : permet de consulter l'objet

2.1.11.3 La multi-collectivité

Les collectivités peuvent être de niveau 1 ou de niveau 2. Les utilisateurs de chaque collectivité héritent de ce niveau. Les utilisateurs de niveau 1 n'ont accès qu'à leur collectivité tandis que les utilisateurs de niveau 2 ont accès à toutes les collectivités disponibles. Lors de la conception de la base de données un champ om_collectivite peut être ajouté à chaque table ayant besoin d'un filtrage par collectivité. Les utilisateurs de niveau 1 ne verront aucune notion de collectivité et n'auront accès qu'aux éléments liés à leur propre collectivité.

2.1.11.4 Les login et logout

Le login se fait par le script `scr/login.php`

`login.php` valorise les variables sessions permettant la gestion des accès et sécurités

```
<?php
  $_SESSION['profil'] = $profil;
  $_SESSION['nom'] = $nom;
  $_SESSION['login'] = $login;
?>
```

La déconnexion se fait avec le script `scr/logout.php`

Le changement de mot de passe se fait avec le script `scr/password.php`

L'accès au changement de mot de passe se fait par défaut dans le menu haut (voir framework/paramétrage)

2.1.11.5 Les utilitaires

La gestion des droits d'accès se fait dans les méthodes des utilitaires :
– `php/openmairie/om_application.class.php` (composant openMairie)
– `obj/utils.class.php`
(*voir framework/utilitaire*)

2.1.12 Tableau de bord et widgets

2.1.12.1 principe

Il est proposé dans ce chapitre de décrire le tableau de bord paramètrable pour les utilisateurs

paramétrage

l'accès au tableau de bord paramètrable dyn/dashboard.inc.php

(voir framework/paramétrage)

Par défaut, le tableau de bord paramétrable est activé, il peut être déconnecté en enlevant le commentaire // die().

les widgets

Les widgets sont des liens et/ou de petits scripts paramétrables qui peuvent être rajoutés dans le tableau de bord. Ces scripts sont conservés dans la table om_widget.

Chaque utilisateur paramètre son tableau de bord.

le tableau de bord paramètrable

Chaque utilisateur choisit ses widgets parmi ceux proposés dans l'application par l'administrateur. Il peut placer ses widgets où il veut dans son tableau de bord. Le paramétrage est conservé dans la table om_tdb

2.1.12.2 widget

Le widget est un petit script qui s'exécute dans le tableau de bord dans une fenêtre normalisée.

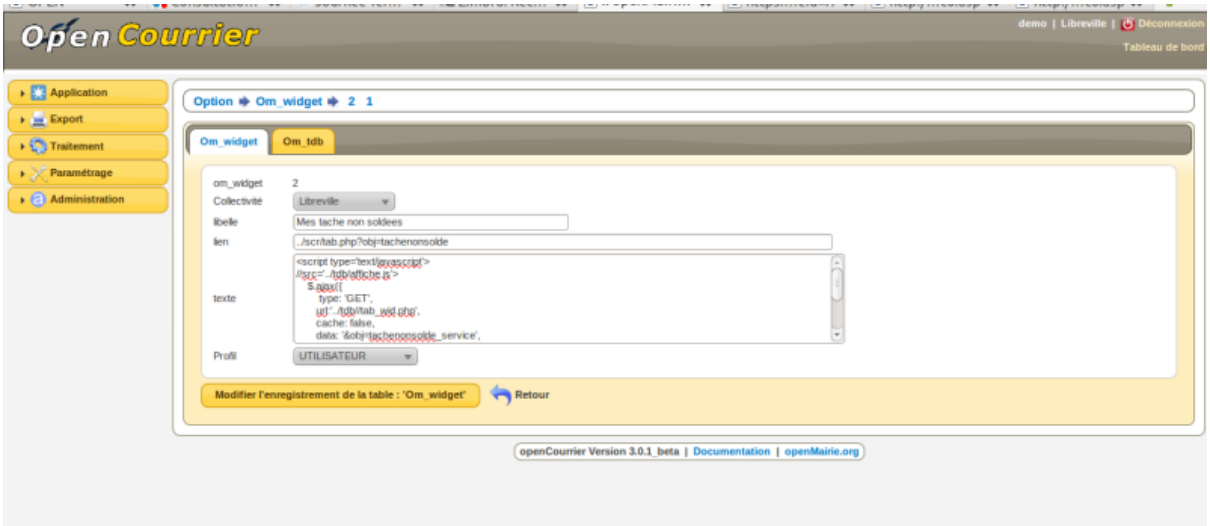
Le script peut faire appel à l'application en cours ou a une application externe. Nous avons mis quelques exemples dans les deux derniers paragraphes. Il est proposé d'abord de vous aider à créer les widgets.

la création de widget

La saisie des widget se fait dans administration -> om_widget.

La grille de saisie est la suivante

```
libellé du widget qui apparaitra à l adition du widget dans le tableau de bord
lien qui sera implémenté (# : pas de lien)
texte : texte du widget (iframe, javascript, ajax ...)
profil : profil autorisé pour le tableau de bord
```



Le tableau de bord, peut gérer toutes sortes d'informations internes ou externes à l'application

```
les taches non soldees pour openCourrier
les appels à la maintenance
l'horoscope, la météo, une vidéo, des photos ...
```

Les widgets internes

les liens sur les cartes (à mettre danbs le champ lien) :

```
la carte de raphele avec tab_sig_point.php
../scr/tab_sig_point_db.php?obj=raphele_1&zoom=6
celle de mas thibert :
../scr/tab_sig_point.php?obj=odp_6&zoom=7
```

les accès personnalisés "ajax"au travers de son code utilisateur (dans openCourrier)

```
<script type='text/javascript'>
    $.ajax({
        type: 'GET',
       url:'../app/tab_wid.php',
       cache: false,
       data: '&obj=tachenonsolde_service',
        success: function(html){
            $('#aff3').append(html);
        }
    });
</script>
<div id='aff3'></div>
```

Ce code lance dans le widget ../app/tab_wid.php ?obj=tachenonsolde_service

tachenonsolde_service est initialisé dans sql/mysql/tachenonsolde_service.inc

Il ne s'affichera que la première page (paramétrer $serie pour le nombre d'enregistrement affichés)

Attention si vous affichez plusieurs widgets "openmairie", mettre un id different pour chaque div (ici aff3)

Les widgets externes

Les autres applications openMairie peuvent aussi être accessibles par widget de la même manière que le paragraphe ci dessus.

D'autres widgets externes sont accessibles en mettant dans le champ texte les scripts suivants :

Acces à une video externe avec un “iframe”

```
<iframe width='200' height='150'
    src='http://www.youtube.com/embed/gS5B4LlqkfI'
    frameborder='0' allowfullscreen>
</iframe>
```

La meteo grace à un javascript du site tameteo.com

```
<div id='cont_f5089b722555454d1872b91f52beafd4'>
    <h2 id='h_f5089b722555454d1872b91f52beafd4'>
    <a href='http://www.tameteo.com/' title='Météo'>Météo</a></h2>
    <a id='a_f5089b722555454d1872b91f52beafd4'
        href='http://www.tameteo.com/
                    meteo_Arles-Europe-France-Bouches+du+Rhone--1-25772.html'
        target='_blank' title='Météo Arles'
        style='color:#666666;font-family:1;font-size:14px;'></a>
    <script type='text/javascript'
        src='http://www.tameteo.com/wid_loader/f5089b722555454d1872b91f52beafd4'>
    </script>
</div>
```

Horoscope au travers d un iframe qui pointe sr astroo.com

```
<!--DEBUT CODE ASTROO-->
<!--debut code perso-->
<iframe width='232' height='302' marginheight='0' marginwidth='0' frameborder='0'
    align='center' src='http://www.astroo.com/horoscope.htm'
    name='astroo' allowtransparency='true'>
<!--fin code perso-->
<a href='http://www.astroo.com/horoscope.php' target='_top'
    title='Cliquez-ici pour afficher l'horoscope quotidien'>
    <font face='Verdana' size='2'><b>afficher l'horoscope du jour</b>
    </font></a>
</iframe>
<noscript>
<a href='http://www.astroo.com/horoscope.php' target='_blank'>horoscope</a>
</noscript>
<!--FIN CODE ASTROO-->
```

Acces à un fil rss avec un module ajax google

```
<script src='http://www.gmodules.com/ig/ifr?url=
   http://www.ajaxgaier.com/iGoogle/rss-reader%2B.xml
   &up_title=Actualit%C3%A9s%20atReal
   &up_feed=http%3A%2F%2Fwww.atreal.fr%2Fatreal%2Fcommunaute%2Factualites-atreal%2FRSS
   &up_contentnr=9&up_fontsize=9&up_lineheight=70
   &up_titlelink=&up_bullet=1
   &up_reload_feed=0&up_reload_fqcy=0
   &up_hl_background=FFFFFF&synd=open&w=200&h=100
   &title=
   &border=%23ffffff%7C3px%2C1px+solid+%23999999&output=js'>
</script>
```

Affichage de photos avec flick ‘r (appel javascript) :

```
<table><tr>
<div class='flick_r'>
<script type='text/javascript'
    src='http://www.flickr.com/badge_code_v2.gne?count=3
        &display=latest&size=s
        &layout=h&source=user
        &user=27995901%40N03'></script>
```

```
</div>
</tr></table>
```

2.1.12.3 le tableau de bord paramétrable

ce paragraphe propose de décrire l'utilisation du tableau de bord paramétrable par utilisateur

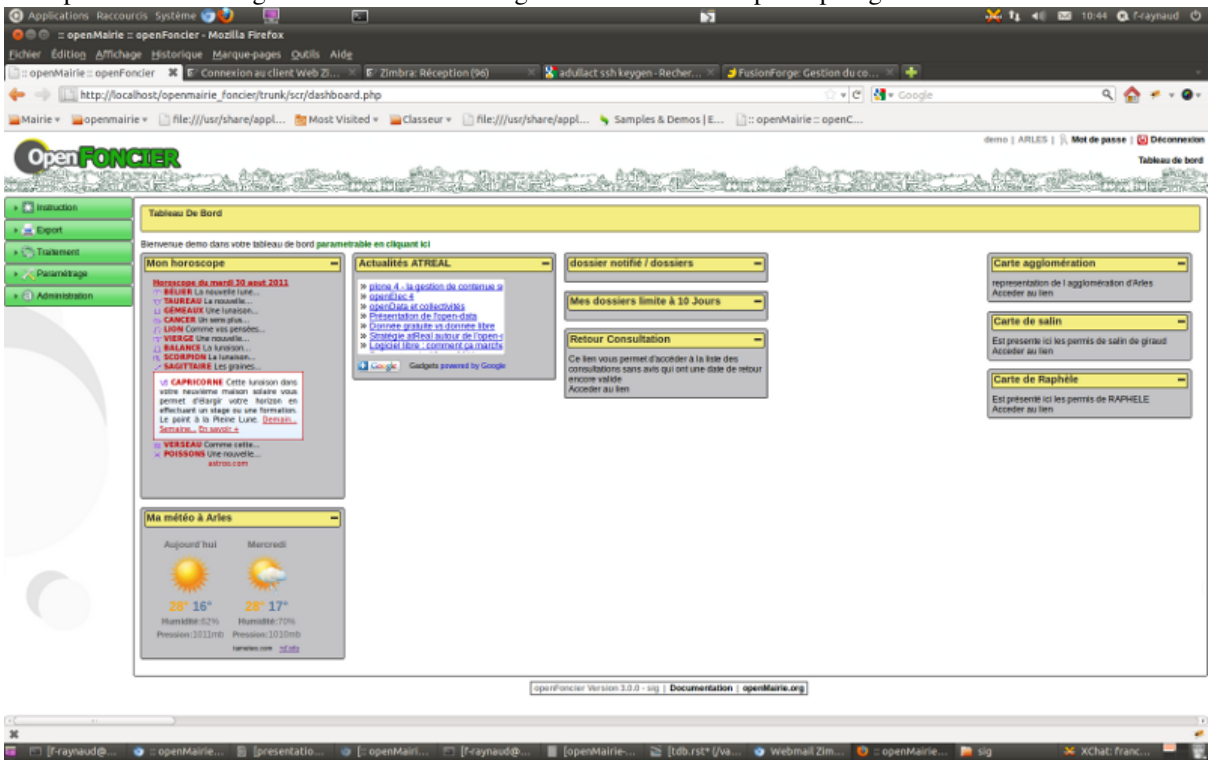
accès au tableau de bord

Le paramétrage se fait en cliquant sur le lien "paramétrer son tableau de bord"

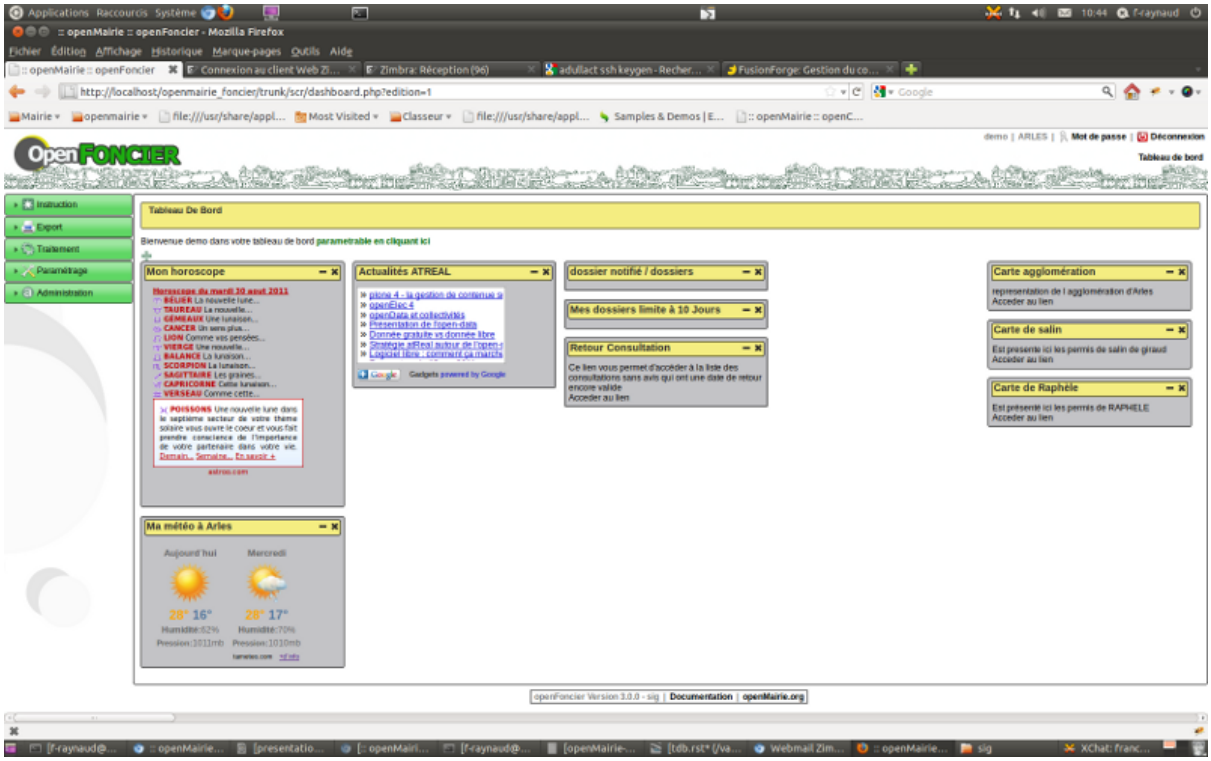
Il apparait alors

```
un "plus"  pour ajouter un widget pour une colone
une croix pour supprimer un widget
```

Le déplacement du widget de haut en bas ou de gauche à droite se fait par copier/glisser avec la souris.



En cliquant sur "+", il est possible de rajouter des widgets dans son tableau de bord



la table om_tdb

La table om_tbd comprend les champs suivants

```
om_tdb int(8) NOT NULL,  : numero d ordre
login varchar(40) NOT NULL, : login de l'utilisateur
bloc varchar(10) NOT NULL, : bloc ou colone (c1 ou c2 ou c3)
position int(8),   : position dans la colone
om_widget int(8) NOT NULL, : numero de widget dans om_widget
```

Attention, en cas de changement de login, un utilisateur perd ses paramètres

2.1.13 L'ergonomie

Depuis la version openMairie 4, il est utilisé l'ergonomie de jquery.

2.1.13.1 Le composant jquery

Les skins jquery peuvent être rajoutés dans le répertoire /om_theme/.

Il est possible de générer un nouveau thème directement depuis le site de Jquery-UI

```
http://jqueryui.com/download    pour télécharger un thème standard
http://jqueryui.com/themeroller pour créer un thème personnalisé
```

Le changement de thème peut se faire dans le fichier EXTERNALS.txt

voir framework/paramétrage

2.1.13.2 Les feuilles de style

Les feuilles de style sont stockées dans le repertoire css/ et sont cascadables

```
main.css        : feuille de style principale openMairie
om-theme/om.css : suivant la feuille de style jquery (voir EXTERNALS.txt)
app/css         : surcharge spécifique a l'application (exemple : le logo de
                  l'application)
```

2.1.14 Les scripts specifiques de l'application

Les méthodes spécifiques à l'application sont dans obj/utils.class.php qui héritent de la class om_application.class.php d 'openmairie

Vous pouvez surcharger les classes d'om_application.class.php dans utils.class.php

Exemple : surcharge de la méthode login() pour conserver le service d'un utilisateur en variable session dans openCourrier.

Ces classes contiennent les méthodes utilisées par le framework mais qui peuvent vous aider à développer les scripts complémentaires de votre application.

Les scripts complémentaires sont mis en répertoire app / et peuvent être créer pour :
– faire un traitement (remise à 0 d'un registre, archivage, export ....)
– faire un sous programme spécifique appellé par un formulaire (bible.php dans openCourrier)
– faire une recherche avec un affichage particulier
Les scripts javascripts sont mis dans le fichier app/js/script.js

Les images spécifiques sont stockées dans app/img

2.1.14.1 Réaliser un script complementaire

Il est proposé ici de vous montrer comment réaliser ce script complémentaire

Le script commence obligatoirement par un appel à la bibliothèque utils.class.php et la creation d un objet $f :

```
require_once "../obj/utils.class.php";
$f = new utils(NULL,
             "courrier",
             _("recherche"),
             "ico_recherche.png",
             "recherche");
```

Les parametres de l'objet sont les suivants :

flag : si flag= Null affichage complete

nonhtml : pas d affichage

htmlonly : tout les elements externes html avec body vide

right : droit géré en om_droit - vide ne verifie pas

title : titre affiché

icon : icone affiché

help : aide affiché

utils.class.php fait la Verification si l utilisateur est authentifié et si l utilisateur a le droit (util.class surcharge core/om_application.class.php qui contient les scripts de base du framework)

Si le paramétre "right" est vide vous pouvez faire appel aux méthodes suivantes

```
isAccredited() // a le droit ou pas
isAuthentified // si non authentifié, il est rejeté

$f->setRight($obj); // affecte un droit d acces
$f->isAuthorized(); //verification que l utilisateur accéde

// Affectation des variables en dehors du constructeur
$f->setTitle($ent);
$f->setIcon($ico);
$f->setHelp($obj);
$f->setFlag(NULL);

// affichage
$f->display();
```

Pour **executer une requête dans un fichier sql** vous devez stocker votre requête dans le répertoire sql/type_de_sgbd/nom_de_requete.inc afin de préserver la portabilité de vos travaux sur d'autres sgbd :

```
// appel au fichier requête
include ("../sql/".$f->phptype."/courrier_scr.inc");

// lancement de la requete sql_courrier et test erreur
$res=$f->db->query($sql_courrier);
$f->isDatabaseError($res);
```

Pour **parcourir les enregistrements** vous utilisez les méthodes dbpear suivantes :

```
// du debut à la fin de la requête
while ($row=& $res->fetchRow(DB_FETCHMODE_ASSOC)){
    // j'affiche le champ courrier
     echo $row['courrier'];

}
```

Pour **ecrire dans la base** vous pouvez utiliser les méthodes insert ou update mais vous pouvez utilisez la méthode autoexecute spécifique à db pear :

requête sql

```
$sql = "INSERT INTO ... ";

$res2 = $f -> db -> query($sql);

$f->isDatabaseError($res2);
```

ou avec un tableau $valF

```
$obj = table

$valF[$obj]=$f-> db -> nextId(DB_PREFIXE.$obj);

$res1= $f-> db -> autoExecute(DB_PREFIXE.$obj,$valF,DB_AUTOQUERY_INSERT);

$f->isDatabaseError($res1);
```

Vous pouvez faire une **Description du role de la page** de la manière suivante

```
$description = _("Cette page vous permet de .. ");

$f->displayDescription($description);
```

Un **message d erreur** s'affiche suivant :

$class : qui est la classe css qui s'affiche sur l'element et qui peut être

"error" : pour le message erreur
"valid" : pour le message de validation

le *code* est le suivant

```
$message = _("Mot de passe actuel incorrect");
$f->displayMessage($class, $message);
```

Pour afficher un **fieldset**, le code est le suivant

```
echo "<fieldset class=\"cadre ui-corner-all ui-widget-content\">\n";

echo "\t<legend class=\"ui-corner-all ui-widget-content ui-state-active\">";

echo _("Courrier")."</legend>";
    ...
echo "</fieldset>
```

il peut être par défaut *ouvert*

```
echo "<fieldset class= ... collapsible\">\n";
```

ou il peut être *fermé*

```
echo "<fieldset ... startClosed\">\n";
```

Vous pouvez faire **appel a des scripts js complementaires** en utilisant la méthode

```
$f->addHTMLHeadJs(array("../js/formulairedyn.js", "../js/onglet.js"));
```

Pour la **gestion des accents**, il est conseillé de ne pas mettre d accent dans le code (utf8 au lieu de latin1-iso8859-1) et de mettre les accents dans la traduction

Pour définir le chemin par défaut pour l' ** upload de fichier**, il faut utiliser la méthode

```
$path=$f->getPathFolderTrs()
```

2.1.14.2 Exemple

Il est proposé de prendre l'exemple du traitement de la remise du registre a 0 dans openCourrier

```
// ENTETE NORMALISEE

/**
 * Cette page permet de remettre a 0 le registre
 *
 * @package openmairie_exemple
 * @version SVN : $Id: xxxx.php 311 2010-12-06 11:43:36 xxxxx $
 */


// CREATION DE L' OBJET $f

require_once "../obj/utils.class.php";
$f = new utils(NULL, "traitement", _("remise a 0 du registre"), "ico_registre.png", "recherche");



// get
if (isset ($_GET['validation'])){
   $validation=$_GET['validation'];
}else{
   $validation=0;
```

```
}


/**
 * Description de la page
 */

$description = _("Cette page vous permet de remettre a 0 le numero de registre ".
                 "Ce traitement est a faire en debut d annee.");
$f->displayDescription($description);


// TEST VALIDATION
// SI = 0 affichage du numero de registre
// SI = 1 mise à 0 du registre et affichage du résultat

if($validation==0){
    $validation=1;

    // REQUETE DU REGISTRE

    $sql= "select id from registre_seq" ;
    $res1=$f->db->getOne($sql);
    $f->isDatabaseError($res1);

    // AFFICHAGE DANS UN FIELDSET

    echo "<fieldset class=\"cadre ui-corner-all ui-widget-content\">\n";
    echo "\t<legend class=\"ui-corner-all ui-widget-content ui-state-active\">";
    echo _("Registre ")."</legend>";
    if ($res1!=0){
        echo "<br>"._("le dernier no du registre est")." :   ".$res1."  ";
    }else{
        echo "<br>"._("vous avez deja fait une remise a 0")."<br>";
    }
    echo "<form method=\"POST\" action=\"num_registre.php?validation=".
    $validation."\" name=f1>";
    echo "</fieldset>";

    // BOUTON DE VALIDATION
    echo "\t<div class=\"formControls\">";
    echo "<input type='submit' value='"._("remise a 0 du registre").
          " ' >";
    echo "</div>";
    echo "</form>";

}else { // validation=1

    // VALORISATION DE $valF
    $valF=array();
    $valF['id']=0;

    // REQUETE MISE A JOUR avec autoExecute
    $res2= $f->db->autoExecute("registre_seq",$valF,DB_AUTOQUERY_UPDATE);
    $f->isDatabaseError($res2);

    // AFFICHAGE DU RESULTAT AVEC UN FIELDSET
    echo "<fieldset class=\"cadre ui-corner-all ui-widget-content\">\n";
    echo "\t<legend class=\"ui-corner-all ui-widget-content ui-state-active\">";
    echo _("Registre ")."</legend>";
    echo "<center><b>"._("remise a 0 du registre reussie")."</b></center>";
    echo "</fieldset>";
```

```
}//validation
```

Notes

_(“Registre ”) : _(“texte”) permet l’utilisation de poedit pour la traduction de texte

class=”cadre ui-corner-all ui-widget-content” : suivant css de jquery

2.1.15 Importer des données en csv

Il est possible d’importer des données suivant des scripts pré-paramétrés mais qui sont modifiables. Pour lancer le menu import, prenez l’option : *administration -> import*.

import_script.php permet les imports dans la base de donnees de fichier au format csv, au moyen d’un formulaire de chargement de fichier.

Exemple de format de fichier à importer (utilisateur.txt) :

```
nom,login,pwd,profil
"Georges DANDIN";"Georges";"21232f297a57a5a743894a0e4a801fc3";"3"
"Raymond DAVOS";"Raymond";"fe01ce2a7fbac8fafaed7c982a04e229";"3"
"Albert DUPONT";"Albert";"05c7e24700502a079cdd88012b5a76d3";"6"
```

La description du transfert se fait dans le fichier extension import_nomobjet.inc dans /sql/... :

exemple : import_script.php ?obj=utilisateur

Dans utilisateur.import .inc , il est defini :

```
le message affiché en import :
    $import= "Insert utilisateur" : Message

la table cible de l'import
    $table= "utilisateur"

la clé primaire si elle est automatique (mise en place d'une séquence) ;
ce champ est vide sinon
    $id="utilisateur"

Le verrouillage de la base de données
    $verrou = 1  mise-à-jour de la base
            = 0 pas de mise-à-jour pour une phase de test

Le mode débug
    $DEBUG  =1 affichage des enregistrements à l'ecran
            =0 pas d'affichage

La mise en place d'un fichier d'erreur :
    $fic_erreur=1 fichier erreur
               =0 pas de fichier d'erreur

La mise en place d'un fichier de rejet reprenant les enregistrements csv rejetés
ce fichier contient les enregistrements en erreur et permet de relancer le
traitement après correction (manuelle)
    $fic_rejet=1 fichier de rejet pour relance traitement
              =0 pas de fichier rejet

La première ligne affiche le nom des champs :
$ligne1=1 la première ligne contient les noms de champs
       =0 sinon

Les zones obligatoires : tableau $obligatoire

    $obligatoire['nom']=1;// obligatoire = 1
```

```
    $obligatoire['login']=1;// obligatoire = 1

les tests d'existence d'une clé secondaire

    $exit['profil']=1 => 0=non / 1=oui
    $sql_exist["profil"]= "sélect profil froc profil cherre profil = '"

La liste des champs à insérer
il faut mettre en commentaire les zones non traitées
    $zone['nom']='0' => la 1ère zone contient le nom
    $zone['login']=1 => la 2ème zone contient le login
    $zone['pwd']='2' => la 3ème zone contient le mot de passe (crypté)
    $zone['profil']='3' => la 4ème zone contient le profil

La valeur par défaut :
En effet, si $zone['profil']="" on peut définir un profil par défaut
    $defaut['profil']='5' Le profil par défaut sera 5
```

2.1.16 Abstraction du 'layout' (ergonomie)

Warning : Cette rubrique est en cours de rédaction.

2.1.17 Abstraction du 'filestorage' (stockage des fichiers)

Warning : Cette rubrique est en cours de rédaction.

2.1.17.1 Principe général

L'objectif de cette rubrique est de gérer le stockage des fichiers dans les applications. Ce stockage permet d'avoir des stockages complexes.

Termes :
- **abstracteur** : classe d'abstraction, c'est elle qui est instanciée et qui instancie les classes spécialisées en fonction des critères,
- **connecteur** : classe spécialisée.

Description des fichiers permettant de gérer le filestorage :

```
[D] dyn
 |-[F] filestorage.inc.php
 |---- ...
[D] core
 |-[F] om_filestorage.class.php
 |-[F] om_filestorage_deprecated.class.php
 |-[F] om_filestorage_filesystem.class.php
 |-[F] om_filestorage_filetransferprotocol.class.php *
 |-[F] om_filestorage_alfresco.class.php *
 |---- ...

* (ce fichier n'existe pas il est là à titre d'illustration de ce que le système est capable de f
```

2.1.17.2 Fonctionnement

Description de l'abstracteur

[core/om_filestorage.class.php]

Ce fichier est composé de deux classes : la classe d'abstraction, la classe de base pour les classes spécialisées.

La classe 'filestorage' est une classe d'abstraction de stockage de fichiers. C'est cette classe qui est instanciée et utilisée par d'autres scripts pour gérer la création, récupération, suppression de fichiers et ce peu importe le stockage utilisé. Son objectif est d'instancier la classe de stockage spécifique aussi appelée plugin de stockage correspondant au paramétrage sélectionné. Cette classe de stockage spécifique hérite de la classe 'filestorage_base' qui lui sert de modèle.

Description du fichier de configuration

[dyn/filestorage.inc.php]

Ce fichier de configuration doit permettre le paramétrage du stockage dans chacune des applications. Par exemple, il permet dans une installation d'openElec de stocker les fichiers en utilisant le connecteur 'filesystem' dans le path /var/openelec/data sur le système de fichiers et dans une autre installation d'openElec de stocker les fichiers dans 'alfresco' sur un autre serveur.

Il doit permettre de sélectionner le stockage à utiliser ainsi que le paramétrage spécifique à chacun des connecteurs.

Par exemple : pour le stockage 'filesystem', il doit être possible de paramétrer le répertoire dans lequel vont être stockés les fichiers, pour le stockage 'alfresco' il doit être possible de paramétrer l'adresse de l'hôte, l'identifiant et le mot de passe de connexion, ...

Il permet aussi de stocker la configuration du stockage de fichier temporaire.

Ce fichier s'inspire des autres fichiers de configuration (mail.inc.php, directory.inc.php, ...). Il doit contenir un tableau associatif :

```
$filestorage = array();
```

Exemple d'un paramétrage (la clé filestorage-default doit être rajoutée dans une des valeurs du paramétrage du fichier database.inc.php comme l'est mail-default ou directory-default) :

```
$filestorage["filestorage-default"] = array {
    "storage" => "filesystem", // l'attribut storage est obligatoire
    "storage_path" => "", // l'attribut storage_path n'est pas obligatoire
    "path" => "/var/www/openfoncier/data/",
    "temporary" => array(
        "storage" => "filesystem", // l'attribut storage est obligatoire
        "path" => "../tmp/", // le repertoire de stockage
    ...
}
```

Si aucun filestorage n'est paramétré, le filestorage deprecated sera instancié et le filestorage temporaire sera le filesystem.

Description des méthodes de la classe filestorage

La classe 'filestorage' contient des méthodes de gestion des fichiers :

`filestorage.`**`create`**(*$data*, *$metadonnees*, *$mode = "from_content"*)

Permet de créer un fichier sur le filestorage,

$data : contenu du fichier

$metadonnees : tableau contenant la liste des métadonnées ($cle => valeur)

$mode ["from_content", "from_path"] :

–from_content : $data contient le contenu du fichier.
–from_temporary : $data l'uid d'un fichier enregistré sur le filesystem temporary.
–from_path : $data contient le chemin du fichier à enregistrer.
Cette méthode retourne l'UUID du fichier enregistré.

`filestorage.`**`update`**(*$uid*, *$data*, *$metadonnees*, *$mode = "from_content"*)
Permet de mettre à jour un fichier sur le filestorage,

$data : contenu du fichier

$metadonnees : tableau contenant la liste des métadonnées ($cle => valeur)

$mode ["from_content", "from_path"] :
–from_content : $data contient le contenu du fichier.
–from_temporary : $data l'uid d'un fichier enregistré sur le filesystem temporary.
–from_path : $data contient le chemin du fichier à enregistrer.
Cette méthode retourne l'UUID du fichier enregistré.

`filestorage.`**`get`**(*$uid*)
Cette méthode retourne le contenu et les métadonnées d'un fichier en fonction de l'UUID passé en paramètre.

`filestorage.`**`delete`**(*$uid*)
Cette méthode supprime un fichier en fonction de l'UUID passé en paramètre.

`filestorage.`**`create_temporary`**(*$data*, *$metadonnees*, *$mode = "from_content"*)
Permet de créer un fichier sur le filestorage temporaire,

$data : contenu du fichier

$metadonnees : tableau contenant la liste des métadonnées ($cle => valeur)

$mode ["from_content", "from_path"] :
–from_content : utilisation normale de la méthode create(), $data contient le contenu du fichier.
–from_path : $data contient le chemin du fichier à enregistrer.
Cette méthode retourne l'UUID du fichier enregistré temporairement.

`filestorage.`**`get_temporary`**(*$uid*)
Cette méthode retourne le contenu et les métadonnées d'un fichier enregistré temporairement en fonction de l'UUID passé en paramètre.

`filestorage.`**`delete_temporary`**(*$uid*)
Cette méthode supprime un fichier temporaire en fonction de l'UUID passé en paramètre.

L'appel aux méthodes "temporary" se fait sur une instance de filesystem défini dans le paramétrage. Ces méthodes sont implémentés dans la classe de base contrairement aux autres méthodes, elle peuvent toutefois être surchargées dans les classes de connecteurs spécifiques.

Description du connecteur depredacted

[core/om_filestorage_deprecated.class.php]

Cette classe est une classe de stockage spécifique aussi appelée plugin de stockage pour le système d'abstraction de stockage des fichiers. Le principe de ce plugin est de stocker tous les fichiers à plat selon la méthode utilisée avant la création du système de stockage. Ce plugin a été créé uniquement dans un soucis de garder la compatibilité pour les applications existantes.

Description du connecteur filesystem

[core/om_filestorage_filesystem.class.php]

Cette classe est une classe de stockage spécifique aussi appelée plugin de stockage pour le système d'abstraction de stockage des fichiers. Le principe de ce plugin est de stocker tous les fichiers en renommant le fichier avec un UUID (identifiant unique) et en créant une arborescence à deux niveaux. Le premier est composé des deux premiers caractères de l'UUID du fichier et le second niveau des quatre premiers caractères de l'UUID du fichier. Un fichier avec l'extension .info permet de stocker les informations de base du fichier ainsi que des métadonnées.

Schéma du stockage :

```
/repertoire/de/stockage/25/252e/252ece72d4c0f88782d9fd6b99f43dfd

/repertoire/de/stockage/ :
/25/ : Le premier niveau des dossiers contenant les deux premiers caractères de l'uuid du fichier,
/252e/ : Le second niveau des dossiers contenant les 4 premiers caractères de l'uuid du fichier, l
252ece72d4c0f88782d9fd6b99f43dfd : Le fichier est stocké avec pour nom un uuid sans extension, la

252ece72d4c0f88782d9fd6b99f43dfd.info : Les fichiers .info sont là pour stocker les métadonnées de

# trois informations obligatoires (ces commentaires ne doivent pas apparaître dans le fichier .ini

filename="plop.pdf"
mimetype="application/pdf"
size="124541"

# métadonnées supplémentaires facultatives (ces commentaires ne doivent pas apparaître dans le fic

propriete1="valeur1"
propriete2="valeur2"

252ece72d4c0f88782d9fd6b99f43dfd_lock : Les fichiers _lock sont là pour servir de marqueur et indi
```

Exemple d'arborescence de stockage :

```
/repertoire/de/stockage/25/252e/252ece72d4c0f88782d9fd6b99f43dfd
/repertoire/de/stockage/25/252e/252ece72d4c0f88782d9fd6b99f43dfd.info
/repertoire/de/stockage/25/252e/252ece72d4c0f88782d9fd6b99f43dfd_lock
/repertoire/de/stockage/25/252e/252eacd35ef547dab12ded6b99f43dfd
/repertoire/de/stockage/25/252e/252eacd35ef547dab12ded6b99f43dfd.info
/repertoire/de/stockage/25/252e/252eacd35ef547dab12ded6b99f43dfd_lock
/repertoire/de/stockage/12/123a/123aacd35ef547dab12ded6b99f43dfd
/repertoire/de/stockage/12/123a/123aacd35ef547dab12ded6b99f43dfd.info
/repertoire/de/stockage/12/123a/123aacd35ef547dab12ded6b99f43dfd_lock
```

Méthode pour générer les uuid :

```
function generate_uuid($prefix = "") {
    return md5(uniqid($prefix, true));
}
```

Description du connecteur filetransferprotocol

[core/om_filestorage_filetransferprotocol.class.php]

Ce fichier permet de déclarer la classe spécialisée pour stocker les fichiers sur un FTP (ce fichier n'existe pas il est là à titre d'illustration de ce que le système est capable de faire).

Description du connecteur alfresco

[core/om_filestorage_alfresco.class.php]

Ce fichier permet de déclarer la classe spécialisée pour stocker les fichiers sur Alfresco (ce fichier n'existe pas il est là à titre d'illustration de ce que le système est capable de faire).

2.1.17.3 Utilisation

Les méthodes de la classe d'abstraction sont désormais utilisées dans la classe upload et dans les widgets upload file du formulaire.

Il est possible de paramétrer une liste de métadonnées d'un champ upload, certains champs de ce formulaire pouvant contenir certaines informations à ajouter aux informations du fichier uploadé, il est necessaire de créer le fichier lors de la validation du formulaire. Pour ce faire le fichier uploadé sera enregistré temporairement sur le filestorage défini pour les fichiers temporaires puis enregistré sur le filestorage définitif lors de la validation du formulaire.

Hors formulaire la méthode create peut être utlisée de 3 façons :
– en lui passant le chemin du fichier dans $data et avec le mode défini à "from_path"
– en lui passant le contenu du fichier dans $data (fonctionnement existant avant modification)
– en lui passant l'UUID d'un fichier temporaire avec le mode défini à "from_temporary"

Configuration du widget Upload

2.1.17.3.1.1 Contraintes Les contraintes sont à rajouter dans la classe métier de l'objet concerné, dans la méthode setSelect.

Exemple de configuration de l'ajout de contraintes de contrôles de la taille maximale et de l'extension lors de l'upload de fichier :

```
<?php
    $params = array(
        "constraint" => array(
            "size_max" => 2,
            "extension" => ".pdf;.txt;.odt"
        ),
    );
?>
```

La taille maximale est en mo et la liste des extensions est une chaîne de caractères.

2.1.17.3.1.2 Métadonnées Il y a des métadonnées globales et spécifiques.

Les globales sont définies dans [obj/om_db_form.class.php] dans l'attribut $metadata_global.

Exemple de configuration :

```
<?php
     var $metadata_global = array(
         "metadonne1" => "méthodeQuiRetourneLaBonneValeur1",
         "metadonne2" => "méthodeQuiRetourneLaBonneValeur2",
     );
 ?>
```

Les specifiques sont à ajouter en attribut de la classe métier de l'objet concerné.

Exemple de configuration de l'ajout de métadonnées :

```
<?php
     var $metadata = array(
         "champ" => array(
             "metadonne1" => "méthodeQuiRetourneLaBonneValeur1",
             "metadonne2" => "méthodeQuiRetourneLaBonneValeur2",
             ),
         ),
     );
?>
```

Les clés de ces tableaux sont les noms des métadonnées, les valeurs associées sont les noms des méthodes qui retournent les métadonnées.

Récupération du fichier

```
//
$file = $f->storage->get($fic);
```

Scripts permettant de visualiser / d'accéder au fichier

2.1.17.3.3.1 spg/file.php Le script permet de télécharger le fichier.

Le code pour composer le lien vers ce script est le suivant :

```
<?php

   $file_download_link = "../spg/file.php?";
   if ($obj != "" && $champ != "" && $id != "") {
       $file_download_link .= "obj=".$obj."&amp;champ=".$champ."&amp;id=".$id;
   } else {
       $file_download_link .= "uid=".$fic;
   }


?>
```

2.1.17.3.3.2 spg/voir.php Le script permet de visualiser le fichier.

```
<?php

   $file_voir_link = "../spg/voir.php?";
   if ($obj != "" && $champ != "" && $id != "") {
       $file_voir_link .= "obj=".$obj."&amp;champ=".$champ."&amp;id=".$id;
   } else {
       $file_voir_link .= "uid=".$fic;
   }


?>
```

CHAPITRE 3

Le générateur

3.1 Le générateur

Précision sur le vocabulaire utilisé dans cette documentation.

Modèle de données

> *« En informatique, un modèle de données est un modèle qui décrit de façon abstraite comment sont représentées les données dans une organisation métier, un système d'information ou une base de données. »*

- cf. Wikipédia, Article Modèle de données.

Dans la documentation suivante, le terme modèle de données est utilisé pour désigner les classes métier d'openMairie ainsi que les formulaires qu'elles représentent.

Objet

Le mot objet fait référence aux instances des classes d'openMairie et, par extension, aux enregistrements en base de données qui les représentent.

3.1.1 Introduction

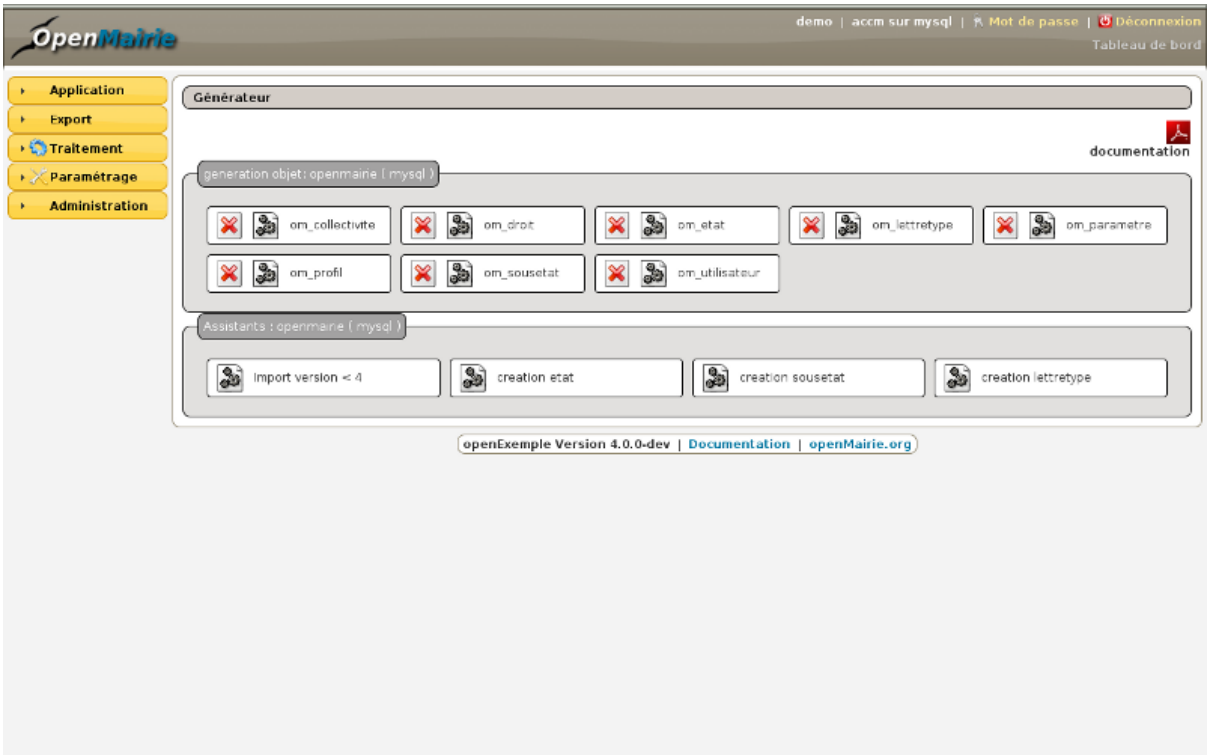
Le générateur permet de construire des applications à partir de l'analyse d'un schéma d'une base de données.

Les informations récupérées dans le schéma sont les suivantes :
– la liste des tables ;
– le nom, le type et les contraintes de chaque colonne.
Sur cette analyse le générateur crée les modèles de données. openMairie gère :
– les clés primaires mono-colonne ;
– les clés étrangères ;
– la multi-collectivité (selon la présence d'un champ `om_collectivite`).
Le générateur contient également des assistants permettant de créer facilement des états associés à des collectivités.

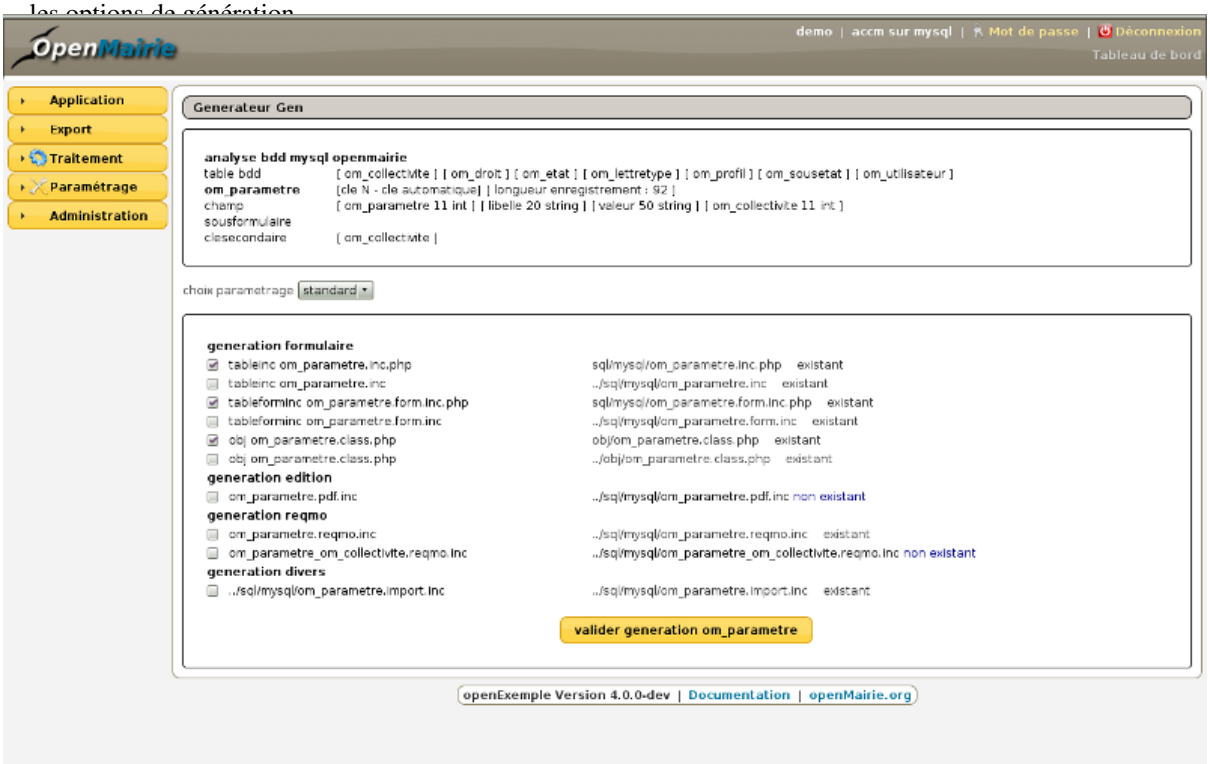
Attention : la version mysql est abandonnée

3.1.2 L'interface

Le menu generateur est le suivant :



En appuyant sur la touche generation on accède à l'écran de génération qui se décompose en :
– une analyse de la base de donnée en cours et de la table choisie
– un état des fichiers existants ou non
– les options de génération


OpenMairie
demo | accm sur mysql | Mot de passe | Déconnexion
Tableau de bord
Application
Export
Traitement
Paramétrage
Administration
Generateur Gen
analyse bdd mysql openmairie
table bdd [om_collectivite] [om_droit] [om_etat] [om_lettretype] [om_profil] [om_sousetat] [om_utilisateur]
om_parametre [cle N - cle automatique] [longueur enregistrement : 92]
champ [om_parametre 11 int] [libelle 20 string] [valeur 50 string] [om_collectivite 11 int]
sousformulaire
clesecondaire [om_collectivite]
choix parametrage standard
generation formulaire
tableinc om_parametre.inc.php sql/mysql/om_parametre.inc.php existant
tableinc om_parametre.inc ../sql/mysql/om_parametre.inc existant
tableforminc om_parametre.form.inc.php sql/mysql/om_parametre.form.inc.php existant
tableforminc om_parametre.form.inc ../sql/mysql/om_parametre.form.inc existant
obj om_parametre.class.php obj/om_parametre.class.php existant
obj om_parametre.class.php ../obj/om_parametre.class.php existant
generation edition
om_parametre.pdf.inc ../sql/mysql/om_parametre.pdf.inc non existant
generation reqmo
om_parametre.reqmo.inc ../sql/mysql/om_parametre.reqmo.inc existant
om_parametre_om_collectivite.reqmo.inc ../sql/mysql/om_parametre_om_collectivite.reqmo.inc non existant
generation divers
../sql/mysql/om_parametre.import.inc ../sql/mysql/om_parametre.import.inc existant
valider generation om_parametre
openExemple Version 4.0.0-dev | Documentation | openMairie.org

3.1.2.1 Analyse de la base

Le programme propose une analyse de la base en cours :
– liste des tables de la base
– l'information sur la clé primaire de la table

– la longueur de l'enregistrement de la table
– les informations sur les champs : nom, type et longueur
– les clés secondaires (exemple table om_colllectivite)
– les sous formulaires à associer
A partir de la version 4.2.0, il n y a plus de choix de paramétrage dans l'écran.

3.1.2.2 Les fichiers à générer

Il est proposé une liste de case à cocher :

La case est cochée sur le fichier correspondant n'existe pas (colonne de droite)

Le formulaire métier auto généré, table.inc, tableform.inc est toujours coché (fichiers en gen/) :
– gen/obj/table.class.php
– gen/sql/basededonnees/table.inc
– gen/sql/basededonnees/table.form.inc
La génération de ces 3 fichiers ne met pas en péril votre programmation qui est en :
– obj/table.class.php
– sql/basededonnees/table.inc
– sql/basededonnees/table.form.inc
basededonnees = mysql ou pgsql

3.1.3 Conditions de génération

3.1.3.1 Contraintes de la base de données

Pour qu'un modèle de données puisse être généré, il faut que la table de base de donneés qui le représente remplisse les conditions suivantes :
– la table doit avoir une clé primaire composée d'une seule colonne, ou une colonne portant le même nom que la table ;
– les clés étrangères doivent référencer des tables remplissant la condition ci-dessus.
Si l'une de ces conditions n'est pas satisfaite, les interfaces de génération affichent une erreur.

3.1.3.2 Contraintes du système de fichiers

Le générateur crée les classes métier de l'application ainsi que les fichiers de surcharge. Pour pouvoir créer ces fichiers, le serveur web PHP doit avoir les droits d'écriture dans les dossiers suivants :
– gen/obj
– gen/sql/pgsql
– gen/sql/mysql
– obj/
– sql/pgsql
– sql/mysql
Si des droits sont manquants :
– les scripts `genauto` et `gensup` ne permettent pas de générer ces fichiers que le serveur ne peut pas écrire ;
– le script `genfull` quant-à lui, affiche des messages après la génération indiquants quels fichiers ne sont pas accessibles (ces fichiers ne sont, bien entendu, pas générés).

3.1.4 Définition des modèles de données

3.1.4.1 L'identifiant

Chaque modèle de données doit avoir un champ destiné à contenir l'identifiant des objets. Sans ce champ, il n'est pas possible de créer un modèle.

Définition de l'identifiant

Il suffit d'ajouter la contrainte SQL `PRIMARY KEY` à une colonne d'une table pour créer un champ identifiant. Il sera ensuite automatiquement géré par openMairie lors de l'ajout, la modification et la suppression d'enregistrements.

Fonctionnement interne du générateur

Comment ce champ est déterminé lors de la génération d'un modèle ?

Le générateur suit la procédure suivante :
– il utilise la colonne ayant la contrainte `PRIMARY KEY` si elle existe ;
– sinon, il utilise la colonne ayant le même nom que la table.
S'il n'existe ni contrainte, ni colonne ayant le même nom que la table, le modèle ne peut pas être créé.

Note : Le fait d'utiliser une colonne ayant le même nom que la table pour déterminer le champ identifiant est là pour une raison de rétro-compatibilité. Les versions d'openMairie antérieures à 4.3.0 n'utilisaient pas encore la contrainte `PRIMARY KEY`.

Est-il possible d'utiliser une clé primaire composée de plusieurs colonnes ?

Non, et c'est bien mieux comme ça.

3.1.4.2 Les références vers d'autres objets

Un modèle de données peut contenir, un ou plusieurs champs, faisant référence à d'autres objets. Ces objets pouvant être de modèle différent.

Définition des références

La méthode pour créer des références diffère en fonction du SGBD.

3.1.4.2.1.1 Avec PostgresSQL Il suffit d'ajouter la contrainte SQL `FOREIGN KEY` à des colonnes pour créer des champs de type référence.

Fonctionnement interne du générateur

Comment ces champs sont déterminés lors de la génération d'un modèle ?

Le générateur conserve une liste des colonnes qui donneront, après génération, des champs références. Pour créer cette liste, il suit la procédure suivante :
– avec PostgresSQL, le générateur peut interroger les tables du système contenant la liste des clés étrangères d'une table particulière, ainsi que les tables étrangères référencées par ces clés ;
– dans un second temps, indifféremment du SGBD, il ajoute a la liste des clés étrangères le nom des colonnes portant le même nom que d'autres tables.
La liste ainsi formée permettra au générateur de créer des champs de type référence dans les modèles de données.

Affichage dans les formulaires

Comment ces champs sont représentés dans les formulaires ?

Depuis le formulaire de l'objet faisant référence, ces champs sont représentés par des balises HTML `<select>`. L'ensemble des objets pouvant être référencés sont listés sous la forme d'options.

Depuis le formulaire de l'objet référencé, un onglet apparaît pour chaque modèle différent faisant référence à cet objet. Chaque onglet liste l'ensemble des objets faisant référence à l'objet présenté en formulaire.

3.1.4.3 Les champs uniques

Un modèle de données peut contenir un ou plusieurs champs uniques. Il n'est pas possible pour plusieurs objets d'un même modèle d'avoir la même valeur pour ce champ.

Un modèle de données généré peut également contenir, au plus, un groupe de champs unique. Cette fois, c'est la combinaison des valeurs de ces champs qui ne pourra exister qu'une seule fois.

Définition des champs uniques

Il suffit de définir une contrainte SQL `UNIQUE` sur une colonne ou un groupe de colonnes pour créer respectivement un ou plusieurs champs uniques.

Fonctionnement interne du générateur

Comment ces champs sont déterminés lors de la génération d'un modèle ?

L'abstracteur de base de données d'openMairie peut, en analysant une table, récupérer la liste de ses colonnes uniques.

Affichage dans les formulaires

Comment ces champs sont représentés dans les formulaires ?

Ces champs sont affichés indifféremment des champs sans contrainte.

Lors de la validation d'un formulaire, une vérification est faite pour chaque champ unique, ainsi que pour un éventuel groupe de champs uniques. Si une valeur (ou combinaison) est déjà présente dans la base de données, un message d'erreur est affiché, et la base de données n'est pas modifiée.

3.1.4.4 Les champs requis

Un modèle de données peut contenir un ou plusieurs champs requis.

Définition des champs requis

Il suffit de définir une contrainte SQL `NOT NULL` sans clause `DEFAULT` sur une colonne pour créer un champ requis.

> **Attention :** En ajoutant une clause `DEFAULT` a une contrainte `NOT NULL` nous indiquons clairement au générateur que **le champ n'est pas requis !** La valeur par défaut permet à l'utilisateur de laisser le champ vide lors d'une validation de formulaire. Le SGBD se charge alors d'ajouter lui même cette valeur.

Fonctionnement interne du générateur

Comment ces champs sont déterminés lors de la génération d'un modèle ?

L'abstracteur de base de données d'openMairie peut, en analysant une table, récupérer la liste de ses colonnes requises n'ayant pas de valeur par défaut.

Affichage dans les formulaires

Comment ces champs sont représentés dans les formulaires ?

Ces champs sont affichés avec un marqueur à côté de leur libellé, indiquant qu'ils sont requis. Par défaut openMairie utilise le caractère * pour indiquer les champs requis.

Si ces champs ne sont pas remplis lors de la validation d'un formulaire, un message d'erreur est affiché pour chaque champ requis non complété, et la base de données n'est pas modifiée.

3.1.4.5 Le champ libellé

Pour représenter des objets dans des champs de type `<select>`, le générateur utilise un champ textuel particulier appelé libellé.

Ce champ est également utilisé pour ordonner les éléments d'un tableau de manière croissante.

Définition du libellé

Pour définir explicitement une colonne comme libellé d'un modèle, il faut la nommer `libelle`.

Si cette colonne n'existe pas, le générateur considère la deuxième colonne de la table comme étant un libellé (ce système était celui utilisé dans les versions d'openMairie 4.2.0 et inférieures).

Enfin s'il n'existe pas de seconde colonne, la clé primaire de la table est utilisé.

3.1.5 Fonctionnalités avancées

3.1.5.1 Ajouter une date de validité à un modèle

Description

Le générateur permet de créer des objets qui seront considérés comme valides seulement pendant une période donnée.

Qu'est ce qu'un objet à date de validité ?

Un objet à date de validité est un objet contenant deux champs spécifiques :
– `om_validite_debut`, déterminant la date de début de validité ;
– `om_validite_fin`, déterminant la date de fin de validité.
Un objet valide se comporte comme un objet « traditionnel ». Par contre, lorsqu'il arrive à expiration, l'objet n'apparaît plus dans les tableaux, les sous-tableaux et les champs de sélection (sauf s'il est actuellement valeur de l'un de ses champs).

Un tel objet est valide lorsque :
– sa date de début de validité est nulle ET (sa date de fin de validité est nulle OU sa date de fin de validité est strictement supérieure à la date actuelle) OU,
– sa date de début de validité est inférieure ou égale à la date actuelle ET (sa date de fin de validité est nulle OU sa date de fin de validité est strictement supérieure à la date actuelle).
A l'inverse, il est considéré comme non-valide (expiré) lorsque :
– il n'est pas valide.
Est-il possible de consulter la liste des objets expirés d'un modèle donné ?

Oui.

Le tableau du modèle dispose d'un bouton `Afficher les éléments expirés` permettant d'afficher, en plus des objets valides, les objets expirés. Lorsque les éléments expirés sont affichés, bouton devient `Masquer les éléments expirés`.

Définition des dates de validité

Pour que le générateur considère un modèle comme « à date de validité », il faut que sa table de base de données contienne les deux colonnes suivantes :
– `om_validite_debut DATE;`
– `om_validite_fin DATE.`

Important : Il ne faut surtout pas définir de contrainte `NULL` ou `DEFAULT` sur ces deux colonnes, sinon ces champs seront obligatoires à chaque validation de formulaire.

Affichage dans les formulaires

Ces champs apparaissent dans les formulaires sous la forme de `datepicker`, comme des champs de type date classiques.

3.1.6 L'analyse de la base

Les informations de la base sont analysées par la méthode « constructeur » de gen.class.php.

La construction des formulaires se fait suivant 5 types de champs reconnus par le générateur :

```
- string : chaîne de caractère
- int : nombre (entier ou décimal)
- date
- blob : texte
- geom : geometry (pour postgres)
```

3.1.6.1 Type de champs

la champ String est du type openMairie (méthode setType()) :
– text dans le cas général
– hiddenstatic si modification pour clé primaire
– select pour clé secondaire
Le champ date est du type openMairie date avec calendrier et java script de contrôle de saisie de date

(La date est au format français JJ/MM/AAAA)

le champ Int est du type openMairie (methode setType())
– hidden si clé primaire en ajout
– hiddenstatic si clé primaire en modification
– text avec contrôle numérique en javascript
– select pour clé secondaire
Le champ Blob est du type openMairie textarea

La longueur et la largeur sont définis en fichier de paramétrage form.inc

La taille n est pas pris en compte dans la longueur d'enregistrement

Les paramètres de dyn/form.inc permettent d'établir la longueur et la largeur d'affichage d'un blob :

```
$max=6; // nombre de ligne blob
$taille=80; // taille du blob
```

Les champs de type geometry sont des champs geom (accès a la fenetre tab_sig.php)

3.1.6.2 Equivalence type pgsql / type openMairie

L'information fournie par postgresql est moins complète que celle de mysql surtout au niveau de la longueur des champs « string » où il est fourni :la longueur de stockage qui est égal à -1 quand le stockage est variable type pgsql (longueur) type tableinfo si different -> type openMairie

```
Bigint      (8)                 int8        -> int
Smallint    (2)                 Int2        -> Int
Integer     (4)                 Int4        -> Int
Real        (4)                 Float4      -> Int
Doubleprecision (8)             Float8      -> Int
Numeric     (20)                Numeric     -> Int
Money       (8)                 Money       -> Int
Char        (1)                 Char        -> String   (Quelque soit la longueur= 1)
Character   (-1)                Bpchar      -> String (Utilisation de la longueur d'affichage)
Character varying (-1)          Varchar     -> String (Utilisation de la longueur d'affichage)
Text        (-1)                text        -> blob  (Utilisation des paramètres de form.inc)
Date        (4)                 Date        -> Date (Utilisation des paramètres de form.inc -
                                               $pgsql_longueur_date)
geometry    -5                              -> geom
boleen                          bolean      -> checkbox
```

Pour postgresql, il est proposé dans form.inc 2 variables qui sont avec la version 4.2.0 inutiles car les longueurs sont gérées par le générateur (valeurs négatives)

```
$pgsql_taille_defaut = 20; // taille du champ par défaut si retour pg_field_prtlen =0
$pgsql_taille_minimum = 10; // taille minimum d affichage d un champ
```

Attention, pour les champs geom, il faut gérer la carte à chercher pour l affichage de la carte en fenêtre

```
exemple de surcharge de la méthode setSelect pour afficher la carte dossier (de la table om_sig_ma

if($maj==1){ //modification
    $contenu=array();
    $contenu[0]=array("dossier",$this->getParameter("idx"));
    $form->setSelect('geom',$contenu);
}
```

3.1.6.3 Nom de champ et nom de table

Attention au nom de tables ou de champs, évitez les termes SQL : match, table, index, type, len ... ou openMairie : objet pour les noms de champs ou table.

3.1.7 Les fichiers générés

Les fichiers générés concernent :
– les formulaires
– les requêtes mémorisées
– le script d'import de données

3.1.7.1 Les formulaires

Les formulaires sont génèrés suivant le nom de la table dans le répertoire sql, sous repertoire portant le nom de la base pour régler le problème de compatibilité SQL (concaténation, extraction ...)

Deux types de formulaire sont générés : type table, type form.

Paramétres de type table

– gen/sql/basededonnees/nom_table.inc
– sql/basededonnees/nom_table.inc

Par défaut :
– tri en affichage vide
– champ de recherche avec les champs string
– pas d'affichage de champ blog
– rattachement de sous formulaire
– affichage de l'édition de la table
Dans le fichier paramètres : form.inc

$serie = nombre d'enregistrement par page

$ico = icône par defaut

Paramétres de type Form

gen/sql/basededonnees/nom_table.form.inc

sql/basededonnees/nom_table.form.inc

Dans le fichier paramètres : form.inc

$ico = icône par defaut

Par défaut :
– tous les champs sont affichés les uns en dessous des autres

3.1.7.2 Les Objets « métier »

L'objet métier généré est stocké en gen/obj/nom_table.class.php. Ce script ne doit pas être modifié car il est reconstitué à chaque génération :

Cela permet de pouvoir modifier la base de données (ajout, modification ou suppression de champs) et de regénérer tout ou partie de l'application

Un second script héritant de l'objet généré permet de surcharger les méthodes et de personnalisé l'objet métier.

Toutes les modifications doivent être faites dans ce script soit en héritant de la méthode, soit en surchargeant la méthode.

L'objet à personnaliser est stocké en obj/nom_table.class.php

Les méthodes générés dans l'objet métier gen/obj/nom_table.class.php sont par défaut les suivantes.

Le type de champs est :
– caché (hidden) en ajout pour la clé primaire automatique,
– modifiable en ajout si la clé primaire n'est pas automatique
– l'unicité de la clé primaire est vérifiée si elle est modifiable (version 4.2.0)
– la clé primaire est visible sans possibilité de modifier en modification
– la clé secondaire n'est pas modifiable en sous formulaire si c'est la clé primaire du formulaire
– la clé secondaire est un champ select qui reprend les informations de la table liée
– la date est au format français
– geom si ce champ est géométrique (version 4.2.0)
La longueur d'affichage et le maximum autorisé à la saisie est celle contenu dans la base d'origine

Le contrôle des clés secondaires des autres tables est généré : il n'est pas possible de supprimer un enregistrement si des enregistrements sont liés à la clé primaire

Il est vérifier l'unicité de la clé si elle n'est pas automatique (version 4.2.0). Les libellés sont les noms des champs.

Ce module sert pour le formulaire et le(s) sous formulaire(s).

Les méthodes qui peuvent être implémentés dans obj/nom_table.class.php sont les suivantes

```
- verifier
- regroupe et groupe pour modifier les présentations [deprecated] utiliser setLayout
- trigger avant ou après l'enregistrement:
- triggerajouter
- triggermodifier
- triggersupprimer
- triggerajouterapres
- triggermodifierapres
- triggersupprimerapres
```

Les méthodes de l'objet généré en gen/obj peuvent être surchargées totalement ou partiellement :

Exemple

```
om_profil.class.php :
    surcharge des méthodes
        setValFAjout setId,
        verifierAjout
        et setType car la clé primaire est numérique et non automatique

om_utilisateur.class.php :
    champ pwd pour mot de passe  methode partiellement surchargées (parent::setvalF($val);)
    setvalF, setType, setValsousformulare,
    surcharge avec un javascript de mise en majuscule du nom
```

Enfin, il est possible de mettre en place d'autres type de champs disponible dans openMairie en surchargeant la méthode setType :

```
- ComboG  combo gauche
- comboD combo droit
- Localisation (geolocalisation en x, y)
- http (lien)
- httpclick (lien)
- Password (Mot de passe)
- Pagehtml (Textearea pour affichage html)
- Textdisabled (Text non modifiable)
- Selectdisabled (Select non modifiable)
- Textreadonly (Text non modifiable)
- Hidden (champ caché)
- Checkbox (case a cocher oui/non)
- Upload (chargement d'un fichier)
- voir (voir un fichier téléchargé)
- Rvb (choisir une couleur rvn avec la Palette de couleur) ...
```

voir framework/formulaire

3.1.7.3 Les états

Seul l'état « pdf » est généré par le générateur

Dans le menu gen (generateur), les états sont générés automatiquement avec un assistant.

Cet assistant vous permet de construire un état :
– en choisissant une table de la base
– en choisissant les champs à mettre dans l'état
L'etat est enregistré dans la table om_etat et peut être modifié menu->administration -> etat

De la même manière, il est possible de créer un sous etat.

Il est possible de choisir le champ qui sera la clé secondaire en lien avec la table mère

Le sousetat est enregistré dans la table om_sousetat et peut être modifié

menu->administration -> sousetat

Le calcul de la largeur des colonnes est automatique dans les sous états et l'état pdf.

Attention : les champs « blob » ne sont pas pris en compte dans les éditions.

3.1.7.4 les requêtes mémorisées

Les requêtes paramétrées sont crées suivant le principe suivant :
– une requête globale
– une requête avec un champ select pour chaque clé secondaire (il est possible de sélectionner la requête à générer)
– Les autres champs sont sélectionnés à l'affichage
Les requêtes sont accessibles dans l'option du menu -> export.

3.1.7.5 les imports

Un script d'import des données est généré suivant le principe suivant :
– si la clé est automatique, génération du compteur
– tous les champs sont importés
– vérification de l'existence de la clé secondaire à chaque enregistrement
Les tables avec clés secondaires doivent donc être importées en dernier.

3.1.8 Paramétrage générateur

Le générateur fonctionne de manière autonome sans fichier de paramétrage. Il est tout de même possible de paramétrer certains éléments grâce à des fichiers de paramétrage déposés dans le dossier `gen/dyn/`.

Il n'est pas nécessaire de personnaliser toutes les variables du fichier. Il est recommandé de déclarer uniquement les paramètres souhaitées. Par défaut, le générateur prend les paramètres inclus dans la classe gen.

3.1.8.1 `gen/dyn/gen.inc.php`

Il permet de définir des paramètres généraux pouvant être utilisés partout dans le générateur.

```
<?php
/***
 * Mode de génération pour la gestion des identifiants et des références
 *
 * Permet de choisir par quel moyen sont récupérées les clés primaires et les
 * clés étrangères :
 *  - "constraints" => en interrogeant les contraintes de la base de données
 *  - "postulate" => par les postulats :
 *    _ "le nom d'un champ 'clé primaire' a pour nom le nom de la table."
 *    _ "le nom d'un champ 'clé étrangère' a pour nom le nom de la table vers
 *      laquelle elle fait référence, et fait référence au champ clé primaire
 *      de cette table."
 *
 * Default : $key_constraints_mode = "constraints";
 ***/
$key_constraints_mode = "postulate";
?>

<?php
/***
 * Liste des tables à ne pas générer
 *
 * Permet de lister les tables dont la génération n'est pas souhaitable. Ces
 * tables n'apparaissent donc plus dans le menu de génération ni dans la
 * génération complète.
```

```
 *
 * Default : $tables_to_avoid = array();
 ***/
$tables_to_avoid = array(
    "om_version",
    "spatial_ref_sys",
);
?>
```

3.1.8.2 gen/dyn/tab.inc.php

Ce fichier de paramétrage permet de lister pour chaque table la liste des colonnes à positionner dans la variable `$champAffiche` lors de la génération des fichiers `gen/sql/<OM_DB_PHPTYPE>/<TAB>.inc.php`.

```
<?php
// Table om_utilisateur
$om_utilisateur = array("nom", "email", "login", "om_profil", );
?>
```

Il est inutile de faire apparaitre ici la colonne portant la contrainte `PRIMARY KEY`. Cette colonne est obligatoire pour le bon fonctionnement du framework.

Il est inutile de faire apparaitre ici la colonne `om_collectivite`. Cette colonne apparaîtra d'office si la colonne est dans la table.

3.1.8.3 gen/dyn/form.inc.php

Voici les paramètres pour la génération de formulaire

```
$serie = 15;                          nombre d'enregistrement par page'
$ico ="../img/ico_application.png"; icone DEPRECATED
$max=6;                               nb de ligne blob
$taille=80;                           taille du blob
$pgsql_longueur_date=12;              taille d'affichage de la date '

*** deprecated
$pgsql_taille_defaut = 20;            taille du champ par defaut si retour pg_field_prtlen =0
$pgsql_taille_minimum    = 10;        taille minimum d affichage d un champ
***/
```

3.1.8.4 gen/dyn/pdf.inc.php

Parametres

```
$longueurtableau= 280;
$orientation='L';// orientation P-> portrait L->paysage";
$format='A4';// format A3 A4 A5;
$police='arial';
$margeleft=10;// marge gauche;
$margetop=5;// marge haut;
$margeright=5;//  marge droite;
$border=1; // 1 ->  bordure 0 -> pas de bordure";
$C1=0;// couleur texte  R";
$C2=0;// couleur texte  V";
$C3=0;// couleur texte  B";
$size=10; //taille POLICE";
$height=4.6; // hauteur ligne tableau ";
$align='L';
// fond 2 couleurs
```

```
$fond=1;// 0- > FOND transparent 1 -> fond";
$C1fond1=234;// couleur fond  R ";
$C2fond1=240;// couleur fond  V ";
$C3fond1=245;// couleur fond  B ";
$C1fond2=255;// couleur fond  R";
$C2fond2=255;// couleur fond  V";
$C3fond2=255;// couleur fond  B";
// spe openelec
$flagsessionliste=0;// 1 - > affichage session liste ou 0 -> pas d'affichage";
// titre
$bordertitre=0; // 1 ->  bordure 0 -> pas de bordure";
$aligntitre='L'; // L,C,R";
$heightitre=10;// hauteur ligne titre";
$grastitre='B';//\$gras='B' -> BOLD OU \$gras='' ";
$fondtitre=0; //0- > FOND transparent 1 -> fond";
$C1titrefond=181;// couleur fond  R";
$C2titrefond=182;// couleur fond  V";
$C3titrefond=188;// couleur fond  B";
$C1titre=75;// couleur texte  R";
$C2titre=79;// couleur texte  V";
$C3titre=81;// couleur texte  B";
$sizetitre=15;
// entete colonne
$flag_entete=1;//entete colonne : 0 -> non affichage , 1 -> affichage";
$fondentete=1;// 0- > FOND transparent 1 -> fond";
$heightentete=10;//hauteur ligne entete colonne";
$C1fondentete=210;// couleur fond  R";
$C2fondentete=216;// couleur fond  V";
$C3fondentete=249;// couleur fond  B";
$C1entetetxt=0;// couleur texte R";
$C2entetetxt=0;// couleur texte V";
$C3entetetxt=0;// couleur texte B";
$C1border=159;// couleur texte  R";
$C2border=160;// couleur texte  V";
$C3border=167;// couleur texte  B";
$bt=1;// border 1ere  et derniere ligne  du tableau par page->0 ou 1";
```

3.1.8.5 gen/dyn/etat.inc.php

parametres

```
$variable='&'; // nouveau
// parametres
$etat['orientation']='P';
$etat['format']='A4';
// footer
$etat['footerfont']='helvetica';
$etat['footerattribut']='I';
$etat['footertaille']='8';
// logo
$etat['logo']='logopdf.png';
$etat['logoleft']='58';
$etat['logotop']='7';
// titre
$etat['titreleft']='41';
$etat['titretop']='36';
$etat['titrelargeur']='130';
$etat['titrehauteur']='10';
$etat['titrefont']='helvetica';
$etat['titreattribut']='B';
$etat['titretaille']='15';
```

```
$etat['titrebordure']='0';
$etat['titrealign']='C';
// corps
$etat['corpsleft']='7';
$etat['corpstop']='57';
$etat['corpslargeur']='195';
$etat['corpshauteur']='5';
$etat['corpsfont']='helvetica';
$etat['corpsattribut']='';
$etat['corpstaille']='10';
$etat['corpsbordure']='0';
$etat['corpsalign']='J';
// sous etat
$etat['se_font']='helvetica';
$etat['se_margeleft']='8';
$etat['se_margetop']='5';
$etat['se_margeright']='5';
$etat['se_couleurtexte']="0-0-0";
```

3.1.8.6 gen/dyn/sousetat.inc.php

parametres :

```
$longueurtableau= 195;
$variable='&'; // nouveau
// parametres

//titre
$sousetat['titrehauteur']=10;
$sousetat['titrefont']='helvetica';
$sousetat['titreattribut']='B';
$sousetat['titretaille']=10;
$sousetat['titrebordure']=0;
$sousetat['titrealign']='L';
$sousetat['titrefond']=0;
$sousetat['titrefondcouleur']="255-255-255";
$sousetat['titretextecouleur']="0-0-0";
// intervalle
$sousetat['intervalle_debut']=0;
$sousetat['intervalle_fin']=5;
// entete
$sousetat['entete_flag']=1;
$sousetat['entete_fond']=1;
$sousetat['entete_hauteur']=7;
$sousetat['entete_fondcouleur']="255-255-255";
$sousetat['entete_textecouleur']="0-0-0";
// tableau
$sousetat['tableau_bordure']=1;
$sousetat['tableau_fontaille']=10;
// bordure
$sousetat['bordure_couleur']="0-0-0";
// sous etat fond
$sousetat['se_fond1']="243-246-246";
$sousetat['se_fond2']="255-255-255";
// cellule
$sousetat['cellule_fond']=1;
$sousetat['cellule_hauteur']=7;
// total
$sousetat['cellule_fond_total']=1;
$sousetat['cellule_fontaille_total']=10;
$sousetat['cellule_hauteur_total']=15;
```

```
$sousetat['cellule_fondcouleur_total']="255-255-255";
// moyenne
$sousetat['cellule_fond_moyenne']=1;
$sousetat['cellule_fontaille_moyenne']=10;
$sousetat['cellule_hauteur_moyenne']=5;
$sousetat['cellule_fondcouleur_moyenne']="212-219-220";
// nombre d enregistrement
$sousetat['cellule_fond_nbr']=1;
$sousetat['cellule_fontaille_nbr']=10;
$sousetat['cellule_hauteur_nbr']=7;
$sousetat['cellule_fondcouleur_nbr']="255-255-255";
```

3.1.8.7 gen/dyn/lettretype.inc.php

parametres

```
// general
$variable='&'; // nouveau
// $variable=chr(163); // compatibilite openmairie <4
// parametres
$lettretype['orientation']='P';
$lettretype['format']='A4';
// logo
$lettretype['logo']='logopdf.png';
$lettretype['logoleft']='58';
$lettretype['logotop']='7';
// titre
$lettretype['titreleft']='41';
$lettretype['titretop']='36';
$lettretype['titrelargeur']='130';
$lettretype['titrehauteur']='10';
$lettretype['titrefont']='helvetica';
$lettretype['titreattribut']='B';
$lettretype['titretaille']='15';
$lettretype['titrebordure']='0';
$lettretype['titrealign']='C';
// corps
$lettretype['corpsleft']='7';
$lettretype['corpstop']='57';
$lettretype['corpslargeur']='195';
$lettretype['corpshauteur']='5';
$lettretype['corpsfont']='helvetica';
$lettretype['corpsattribut']='';
$lettretype['corpstaille']='10';
$lettretype['corpsbordure']='0';
$lettretype['corpsalign']='J';
```

CHAPITRE 4

L'information géographique

4.1 L'information géographique

Il est necessaire que l'API openLayers soit dans le framework :

lib/openlayers

4.1.1 Principe

Il est proposé dans ce chapitre de decrire le module tab_sig.php qui permet la geo localisation d'objet dans openMairie

A partir de la version 4.4.0, l'accès au module géographique se fait en mettant l'option_localisation d'om_paramètre à la valeur "sig_interne"

L'objectif de tab_sig_map est de permettre une saisie le plus souvent automatique par un point, ligne, multiligne, polygone, multipolygone. Cette saisie est stockée dans la base métier postgresql. Elle est affichée sur des fonds existants sur internet : google sat, openStretmap ou bing (pour l instant) en utilisant le composant javascript openLayers

Il n'est donc pas nécessaire de disposer d'un SIG pour utiliser tab_sig.php.

Le format de stockage des données pgsql est celui de l'OGC et il est accessible aux clients libres où propriétaires qui respectent ce format (QGIS, GRASS, VEREMAP ... pour les clients libres)

4.1.1.1 Géo localisation automatique

L'enjeu est de limiter au maximum la géo localisation manuelle dès qu'il y a une possibilité de géo localisation automatique.

Elle se fait au travers de 2 programmes (voir paragraphe sur le geocodage) :
– adresse_postale.php : positionnement suivant le numero et rue
– adresse_postale_google.php : positionnement suivant le numero et rue avec google
– adresse_postale_bing.php : positionnement suivant le numero et rue avec bing
– adresse_postale_mapquest.php : positionnement suivant le numero et rue avec mapquest
La géolocalisation automatique peut se faire sur une base externe postgresql (eventuellement via une vue)

le script tab_sig.php permet de saisir manuellement le point.

4.1.1.2 Affichage de carte

L'affichage se fait avec openLayers dont le composant est de base dans le framework openMairie : lib/openLayers. (le composant est installé de manière a être optimisé avec une css openmairie)

La librairie proj4 inclus dans lib/openLayers permet de pouvoir utiliser les projections lambert sud et lambert 93.

La projection géographique et Mercator est de base dans openLayers

L'enjeu est donc de projeter les données stockées dans la base "métier" postgresql - postgis (les communes devant utiliser le lambert93) en mercator pour être lisible avec les cartes accessibles sur internet.

L'affichage des datas est fait au travers d'une requête postgresql qui alimente un tableau json lu comme une couche openLayers.

La data à modifier est fourni par requete postgresql au format wkt à openLayers. (voir paragraphe layers)

tab_sig.php permet

```
- l affichage de/des  fond(s)
- l'affichage de données (data)
- l affichage du geométries qui peut être créé ou déplacé (couche wkt)
```

dans la version 4.2.0, tab_sig permet aussi

```
- l'affichage de flux wms et wfs (getmap) et de recuperer les données (getfeature)
- la collation de géométrie dans un pannier et son enregistrement en multi géométries
```

4.1.1.3 Paramétrage de la carte

Le paramétrage général (contenu dans scr/tab_sig.php) des cartes est modifiable dans dyn/var_sig.inc

```
// *** parametre de tab_sig.php ***
// generer une cle pour le site : http://code.google.com/intl/fr/apis/maps/signup.html
$cle_google = "";
$fichier_jsons="json_points.php?obj=";
$fichier_wkt="wkt_point.php";
//zoom par couche : zoom standard permettant un passage de zoom a l autre
$zoom_osm_maj=18;
$zoom_osm=14;
$zoom_sat_maj=8;
$zoom_sat=4;
$zoom_bing_maj=8;
$zoom_bing=4;
// popup data contenuHTML
$width_popup=200;
$cadre_popup=1;
$couleurcadre_popup="black";
$fontsize_popup=12;
$couleurtitre_popup="black";
$weightitre_popup="bold";
$fond_popup="yellow";
$opacity_popup="0.7";
// image localisation maj ou consultation
$img_maj="img/punaise.png";
$img_maj_hover="img/punaise_hover.png";
$img_consult="img/punaise_point.png";
$img_consult_hover="img/punaise_point_hover.png";
$img_w=14;
$img_h=32;
$img_click="1.3";// multiplication hauteur et largeur image cliquee

// *** parametres d om_sig_map.class.php, om_sig_wms.class.php
$contenu_etendue[0]= array('4.5868,43.6518,4.6738,43.7018',
                           '4.701,43.3966,4.7636,43.4298',
                           '4.71417,43.64,4.72994,43.65166',
                           '4.72345,43.55348,4.73134,43.55932',
                           '5.2094,43.4136,5.3345,43.4759'
                           );
$contenu_etendue[1]= array('agglomeration',
```

```
                            'salin de giraud',
                            'raphele',
                            'Mas thibert',
                            'vitrolles'
                            );
$contenu_epsg[0] = array("","EPSG:2154","EPSG:27563");
$contenu_epsg[1] = array("choisir la projection",'lambert93','lambertSud');
```

La version 4.4.0 contient les étendues des communes des bouches du rhône.

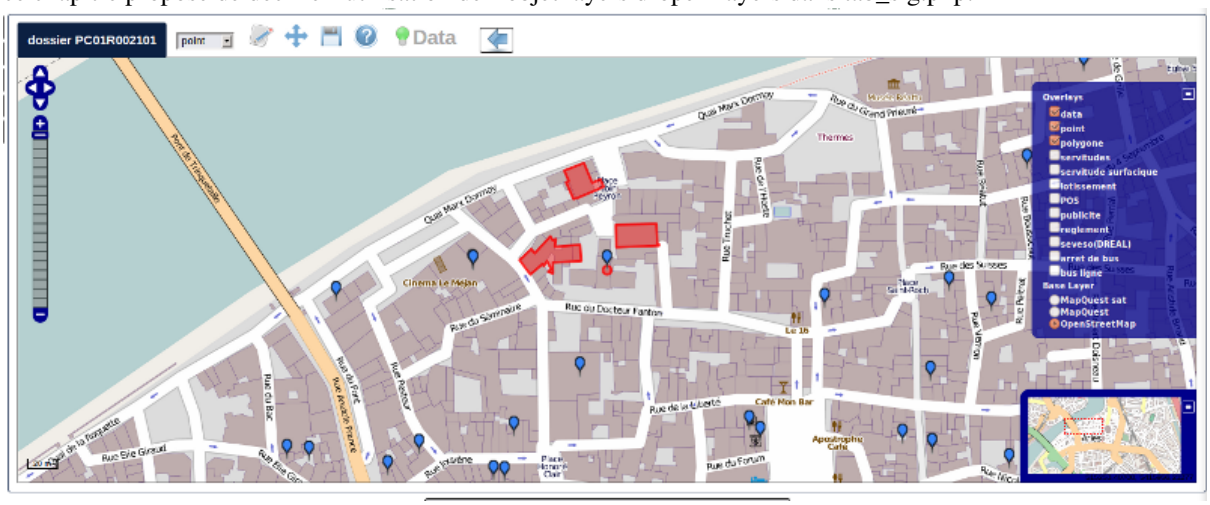
4.1.2 objet map

ce chapitre propose de décrire l'utilisation de l'objet map d'openLayers dans tab_sig.php.

Cet objet permet de définir
– le div ou la carte sera affichée (dans tab_sig.php la carte s'affiche avec le div : map-Id)
– les options de la carte et les controles affichés

4.1.3 afficher les layers

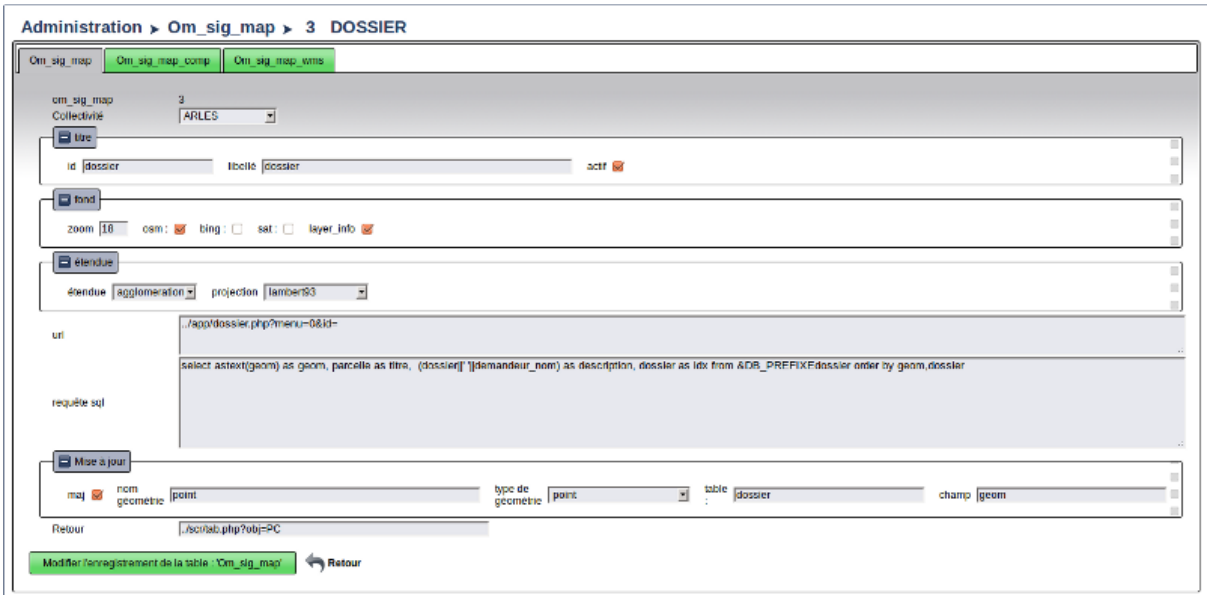
ce chapitre propose de décrire l'utilisation de l'objet layers d'openLayers dans tab_sig.php.



Dans le lien, il est possible de définir

```
- la  carte a afficher suivant l'id : ?obj=   Obligatoire
- le fond affichable par défaut : sat, bing, osm : &fond =
- l'étendue : &etendue =
- l enregistrement à modifier : &idx=
```

Les cartes sont paramétrées dans om_sig_map (menu administration)



Il est possible de copier une carte et de paramétrer les champs suivants :

```
- id : identifiant unique (obligatoire)
- libelle
- fonds a afficher et data (osm, bing, sat(google))
- étendue et epsg (voir sig/var_sig_point.inc)
- url (qui pointe sur la fiche ou le formulaire de saisie)
- requete sql qui affiche les données json et qui doit désigné :
    le titre
    la description
    l idx
- la mise a jour si oui,
            le champ géometrique et la table maj,
            le type de géometrie et le nom de la couche openLayers (version 4.2.0)
- le retour de la carte
```

Dans tab_sig.php, il y a 3 types de layers :
– les fonds de cartes existants sur internet (base layers)
– les données issus de postgresql (overlays)
– les données wms (overlay)

4.1.3.1 Les fonds

Il est proposé les fonds suivants :

osm : openstreetmap

sat : satelite google

bing : satellite microsoft

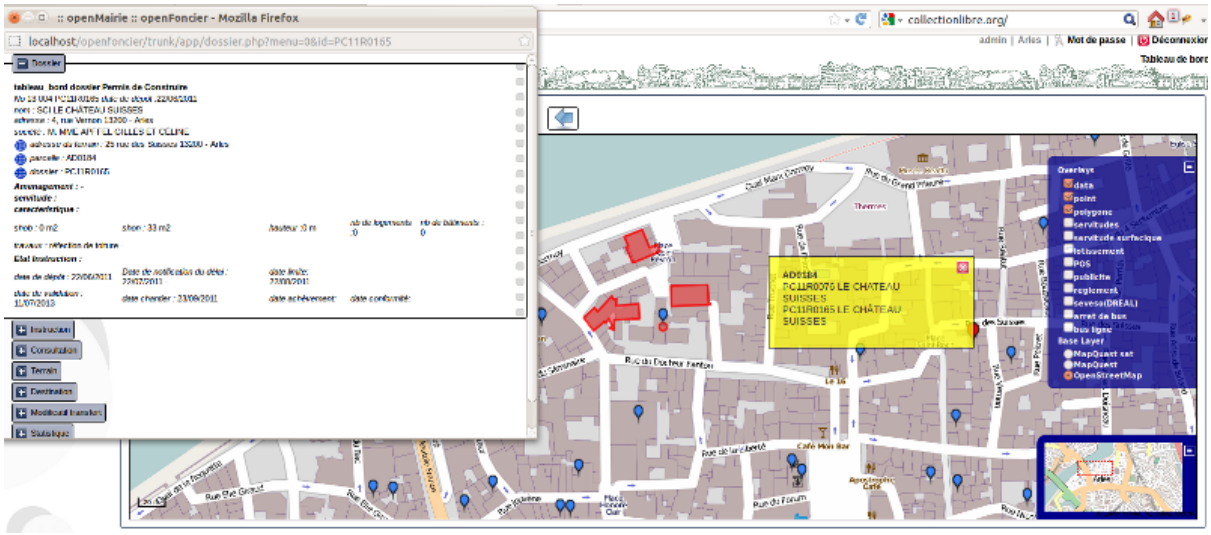
4.1.3.2 Les datas

Information de la carte : layer_info

Cette couche fait appel à sig_json.php

Il est possible de faire appel a un autre script (voir dyn/var_sig.inc)

La requête pgsql est paramétrée dans la table om_sig_map et doit définir les champs geom, titre, description et texte.

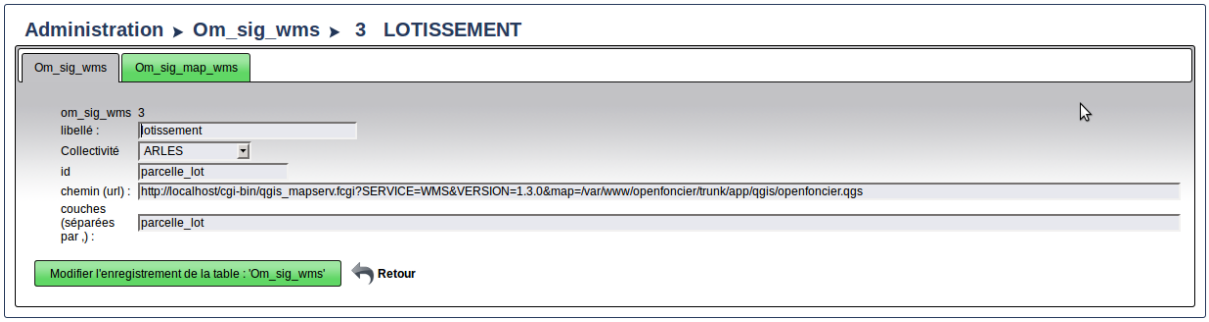


sig_json.php présente tous les enregistrements d'un même point (même géom) sur un seul popup

En effet, il est constitué un popup lorsque l on clique sur l objet et donne la possibilité à un accès URL parametrée dans om_sig_map

4.1.3.3 Les flux wms

Le paramètrage des flux wms est saisi dans om_sig_wms



il faut saisir

```
- libelle du champ
- la collectivité
- l'identifiant (il doit être unique pour chaque couche wms)
- le lien de la couche (http)
- les layers de la couches séparés par une virgule
```

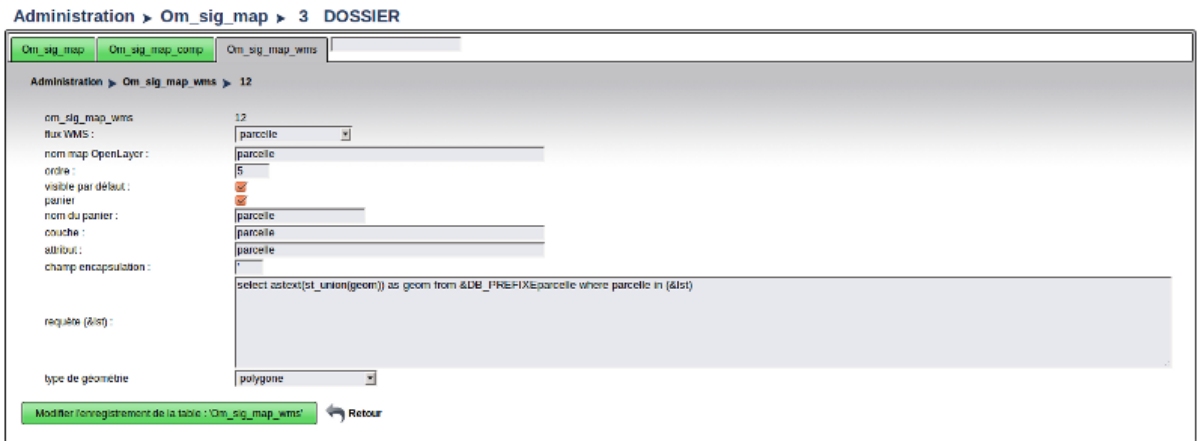
Exemple de lien avec qgis serveur

```
http://localhost/cgi-bin/qgis_mapserv.fcgi
    ?SERVICE=WMS&VERSION=1.3.0
    &map=/var/www/openfoncier/trunk/app/qgis/openfoncier.qgs
```

L'affectation des flux wms dans une carte est saisi dans om_sig_map_wms

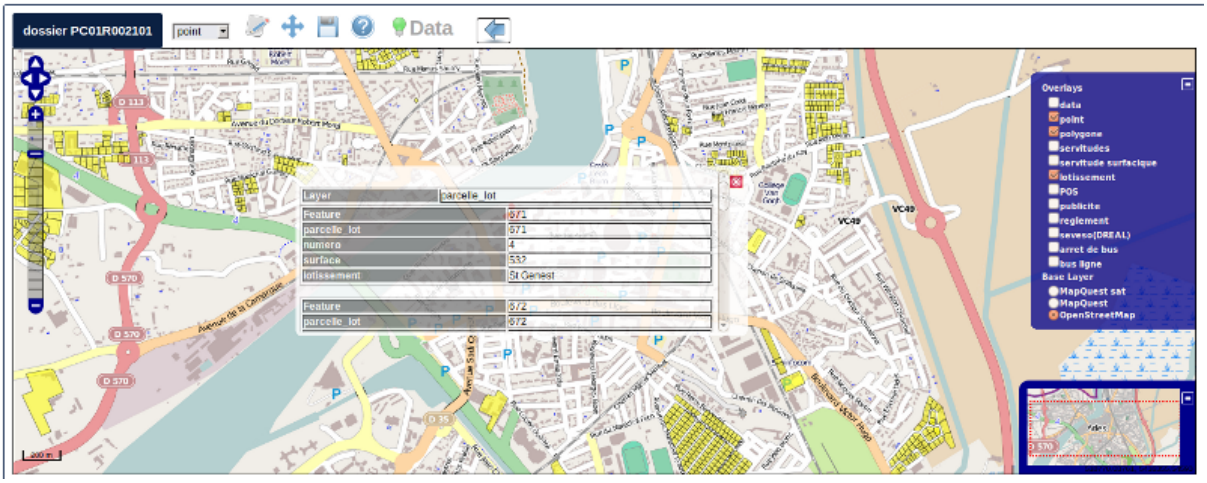
Il est saisi

```
le nom du flux wms
nom du layer sur la carte
l ordre d affichage
la visibilité par défaut (case à cocher)
```



Sur la carte ci dessous le flux wms est activé et affiche le lotissement (getMap)

En cliquant sur le lotissement, il est possible d'accéder aux données (getFeature)



version 4.4.0

Trois nouveaux paramètres sont disponnibles :

sqlfilter : possibilité de filtre du flux wms (attribut FILTER)

compléter la zone avec une requête SQL qui va généré le filtre (syntaxe suivant le serveur WMS)

exemple d'un filtre

```
pour produire le filtre suivant :
layer1:"champ1" = 'valeur1',layer2:"champ2" = 'valeur2'

il faut entrer la requête suivante pour selectionner les electeurs d'un bureau :

select 'electeur:²bureau² = '''||bureau.bureau||''' as buffer from &DB_PREFIXEbureau where bureau


select 'electeur:²bureau² = ''&idx'' as buffer from &DB_PREFIXEbureau where bureau = '&idx'

² = caractère utilisé pour les doubles quotes : "
|| concatenation sql
''' permet d echapper la simple quote
'' sql remplace les deux quotes par une quote (caractere quote)

le filtre final appliqué au flux wms est : electeur:"bureau" = '04'  pour le bureau 04
```

base_layers : possibilité d'utiliser le flux wms comme base layers (au même niveau qu'OSM)

single_tile : raméne le flux wms en une seule image pour la fenêtre et non en imagette (permet de corriger les labels tronqués)

Attention les temps de réponses peuvent s'allonger car il n'y a pas de cache.

4.1.3.4 La notion de pannier

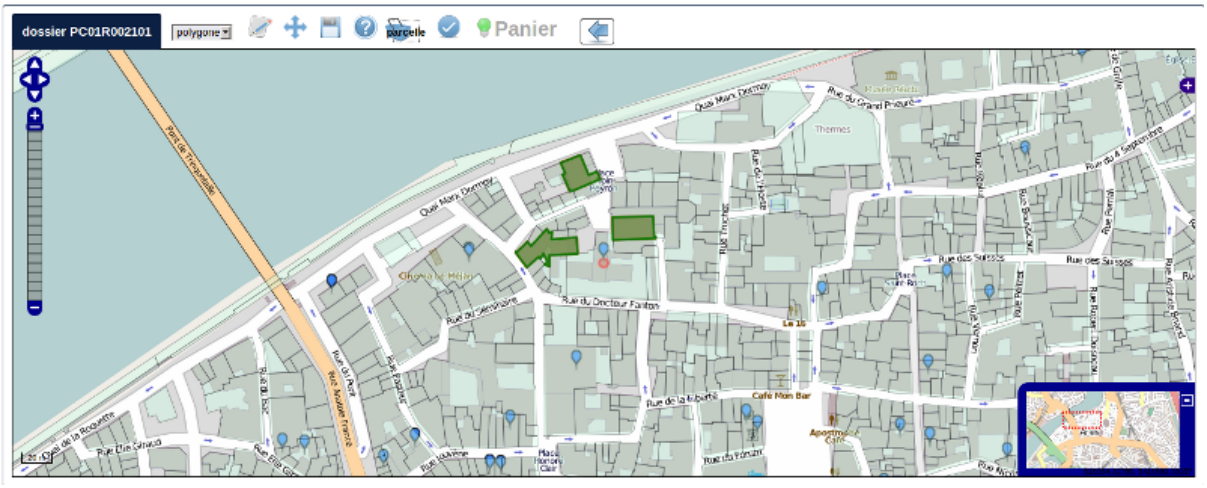
Le pannier permet de pouvoir stocker des géométries au travers de flux wms mais attention, la géométrie est récupérée dans une table ou une vue postgis (c'est pour l'instant une limite de la version 4.2.0)

exemple : openFoncier carte dossier :

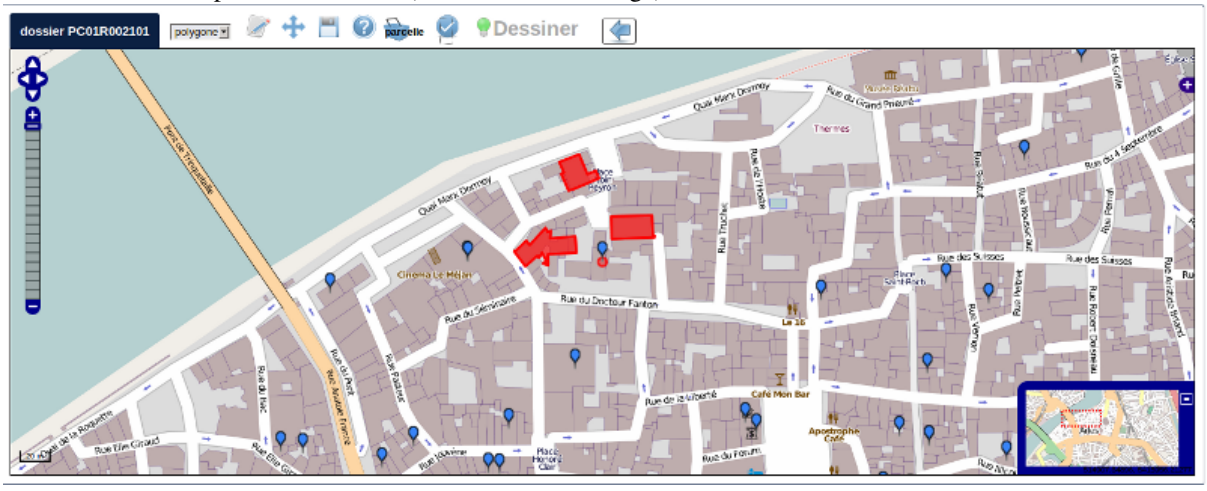
Il est proposé dans ce cas de stocker des polygones dans le pannier et de sauvegarder un multipolygone constitué de ces polygones récupérés dans le pannier

Choisir dans le select "polygone" ; L'etat est "dessinner"

Il apparait le pannier "parcelle". Sélectionner les parcelles en cliquant dessus (elles sont vertes)



Valider une fois les parcelles choisies (elles deviennent rouge)



Appuyer sur "enregistrer", l'état devient enregistrer

Cliquer sur le jeu de parcelles de votre choix (ce jeu devient vert clair)

Il peut y avoir un ou plusieurs panniers : exemple : parcelle, batiment. par contre la géométrie récupérée ne concerne qu une seule couche

la gestion de pannier se fait dans om_sig_map_wms

```
panier :          option pannier activé (Oui/non)          Exemple dossier/openFoncier :
pa_nom :          nom du pannier                           parcelle
pa_layer :        nom du layer pannier                     parcelle
pa_attribut:      attribut de la couche à récupérer        parcelle
pa_encaps:        caractère d'encapsuation (la ')          '
pa_sql:           requête de récupération                  select astext(st_union(geom)) as geom
                                                           from &DB_PREFIXEparcelle where parcelle i
pa_type_geometrie:  type de géométrie                      polygone
```

le script de gestion de pannier est : scr/sig_pannier.php

4.1.3.5 La géométrie à modifier : couche vectors :

Le chargement de la couche vectors se fait si dans la table om_sig_map, la case maj est activée.

La géométrie est récupérée par le script sig_wkt.php (appel a un script paramètrable dans var_sig.inc) et la carte est centrée sur la géométrue

Il est possible de :

- positionner manellement la géométrie
- déplacer la géométrie
- **enregistrer la géometrie** [selectionner la géométrie, le programme] form_sig.php est chargé en fenetre et permet de supprimer la géométrie (champ geometrique = null) ou modifier cette géométrie.

Les fonctions javascript et les controles sont activées suivant chaque état.

Dans dyn/form_sig_update.inc.php, il est possible de paramétrer des post traitements de saisie

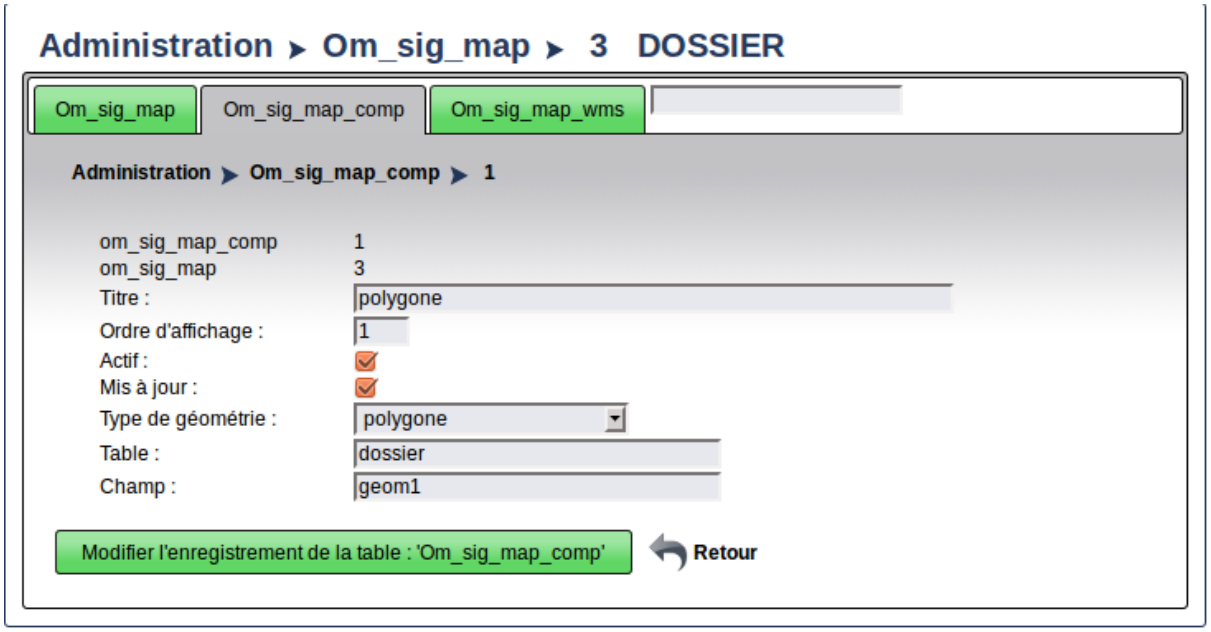
Dans dyn/form_sig_delete.inc.php, il est possible de paramétrer des post traitements de suppression

4.1.3.6 Les géométries complémentaires

Il peut y avoir plusieurs géométries pour un même objet.

Elles sont saisies dans om_sig_map_comp

```
titre                polygone    nom de la nouvelle géométrie
ordre d affichage    1           ordre d'affichage dans le select
actif                coché       activé la nouvelle géométrie
Mise a jour          coché       autorisé la mise à jour
type de géométrie    polygone    polygone, point, ligne
table                dossier     table du champ géométrique
champ                geom1       champ géometrique concerné
```



Dans l exemple précédent, il apparait une fenêtre select ou l utilisateur a le choix entre une géométrie “point” et une géométrie “polygone” du fait de la mise en place d’une géométrie complémentaire.

4.1.4 installation d’om_sig_map

Pour faire fonctionner tab_sig.php, il faut :
– installer postgis sous postgres
– openlayers qui est de base dans le framework lib/openlayers

4.1.4.1 optimisation composant openLayers

construire un OpenLayers.js compresse dans le repertoire build

```
$ cd buill
$ python build.py
```

le fichier fait 800 ko au lieu de 3 Mo
– compression lite

```
  $ python build.py lite.cfg
  le fichier fait 120 ko
  regarder dans le fichier "lite" les fichiers qui sont inclus
  et éventuellement le compléter
```

4.1.5 postgis

ce chapitre propose de décrire les possibilités d’utilisation de postgis dans openMairie.

4.1.5.1 principes

Il est proposé un renvoi sur la documentation française.

http ://postgis.refractions.net/documentation/manual-1.3/ch06.html

ou sur le projet pédagogique postgis (conférence avec documentation et nombreux exemples :

https ://adullact.net/projects/postgis/

Les requêtes utilisant des fonctions potgis peuvent être implémentées dans “reqmo”

Il sera proposé un lien sur un tutorial utilisant les fonctions postgis.

4.1.5.2 base et schéma

Il est noté que les applications openMairie peuvent s’installer dans un schéma.

Les tables et fonctions postgis sont alors accessible dans le schéma public.

4.1.6 geocodage

Ce chapitre est consacré au problème de géocodage.

Il est propose un géocodage interne (sgbd adresse en reseau interne) ou un geocodage externe

Il convient de regarder les termes de licences concernant les API externes non libres (mapquest/osm) afin de s’assurer de bien respecter les obligations de l’autorisation gratuite.

Un document décrivant les contraintes juridiques et techniques de l’utilisation des API est accessible via ce lien.

Le parametrage se fait dans le fichier sig/var_adresse_postale.php

4.1.6.1 var_adresse_postale.inc

paramètrage général

```
$longueurRecherche=1;
```

adresse postale stockée sur une base dans le réseau interne

```
// table et champs de la requete adresse postale ou l information doit etre recupere
$t="adresse_postale";                       // table adresse postale
$t_voie = "rivoli";                         // code adresse
$t_numero="num_voi";                        // numero dans la voie
$t_complement="suf_voi";                    // suffixe (bis, ter ...)
$t_geom="the_geom";                         // geometry point(X,Y)
$t_adresse="(typevoie ||' '||nomvoie)";     // libelle de l adresse
$t_quartier="id_parc";
// *** a voir
$t_cp='';                                   // nom champ cp
$t_ville='';                                // nom champ ville
$t_insee='';                                // nom champ insee

// champ du formulaire ou l adresse est saisi pour retour du point de geolocalisation
$f_numero='numero_voie';        // nom champ du numero dans la voie
$f_voie='voie';                 // nom champ du code de la voie (rivoli)
$f_complement='complement';     // nom champ du complement de numero
$f_geom='geom';                 // nom champ geometrique point(X,Y)
$f_libelle='libelle_voie';      // nom champ libelle de la voie
// *** a voir
$f_cp='';                       // nom champ cp
```

```
$f_ville='';                    // nom champ ville
$f_insee='';                    // nom champ insee
```

Cas ou la table d'adresse est stockées dans une autre base (voir parametrage framework de database.inc.php) :

```
$db_externe='Oui';  //  Oui : base externe
$dsn_externe= array(
    'title'  =>"base locale des adresses de l'IGN",
    'phptype'  => "pgsql",
    'dbsyntax' => "pgsql",
    'username' => "postgres",
    'password' => "postgres",
    'protocol' => "tcp",
    'hostspec' => "localhost",
    'port'     => "5432",
    'socket'   => "",
    'database' => "ignlocal",
    'formatdate'=> "AAAA-MM-JJ",
    'schema'  => "public",
    'prefixe'  =>""
);
$db_option_externe=array('debug'=>2,
    'portability'=>DB_PORTABILITY_ALL);
```

Géolocalisation par accès à un API externe

```
// variables par defaut cp et ville si non renseignées dans le formulaire
// pour recherche
$cp="13200";
$ville="Arles";
$pays = ""; // a voir
// epsg de transformation pt adresse postale dans la base en cours
$epsg= "EPSG:27563";
// acces au script adresse_postale externe
$adresse_interne="Oui";
$google="Oui"; // google
$bing="Oui";   // bing
$osm="Oui";    // mapquest
```

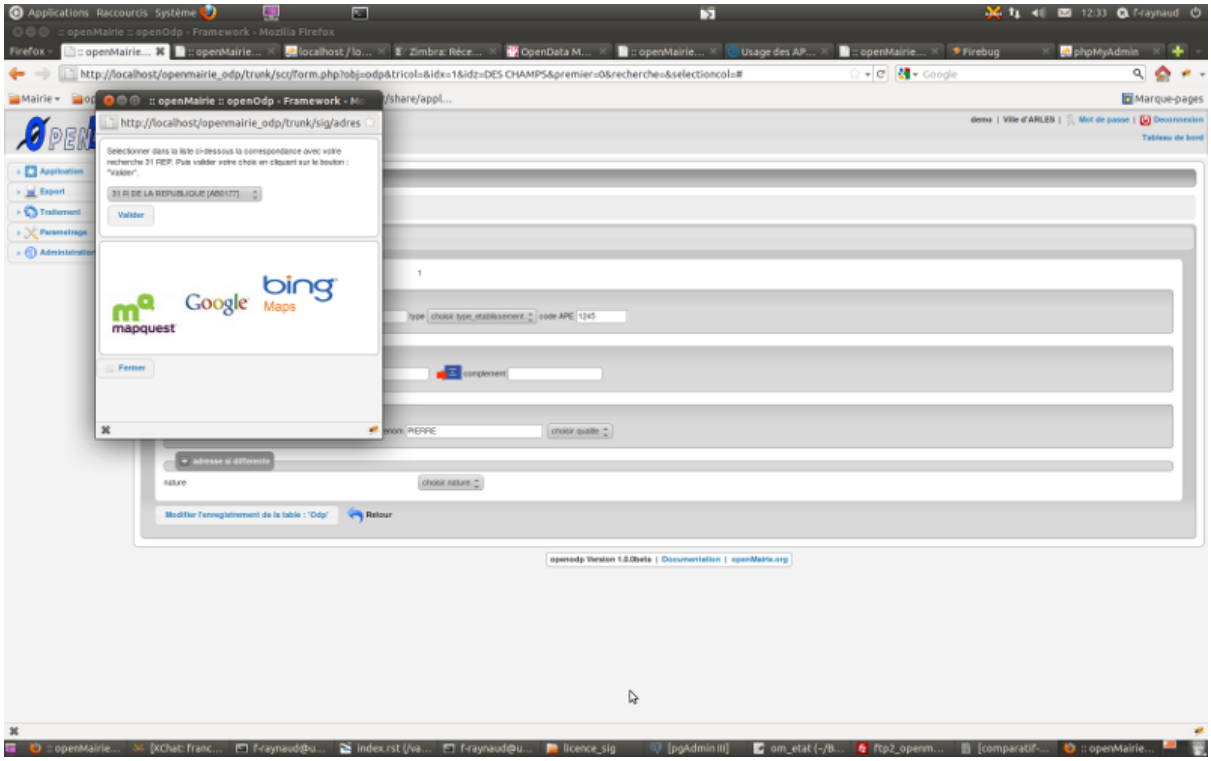
le parametre $adresse_interne à Oui permet de consulter une adresse stiockée dans le réseau interne sur la même base, ou sur une base différente (voir plus haut)

Ensuite 3 API peuvent être initialisés : google, bing et mapquest

4.1.6.2 Mise en oeuvre dans un formulaire d'un bouton de la geolocalisation

La géolocalisation se fait sur la base du script

```
sig/adresse_postale.php
qui fait appel suivant le paramétrage à :
    adresse_postale_bing.php
    adresse_postale_google.php
    adresse_postale_mapquest.php
```



Il est appelé depuis la classe métier suivant l'exemple suivant :

Exemple de openmairie_domainepublic : objet odp

```
dans sql/pgsql/odp.form.inc : le champ adressepostale est implementé comme un champ vide
$champs=array("odp", ...
            "'' as adresse_postale",  // specific

dans obj/odp.class.php

dans la methode setType, le champ adresse_postale est du type httpclick

    function setType (&$form, $maj) {
        parent::setType ($form, $maj);
        $form->setType('adresse_postale', 'httpclick');

avec la methode setVal : valoriser par défaut l'accès au script adresse_postale
                         app/js/script.js

   function setVal(&$form, $maj, $validation, &$db, $DEBUG=null){
       // bouton adresse postale
       $form->setVal("adresse_postale",
        "adresse_postale('f1',f1.libelle_voie.value,f1.numero_voie.value)");
   }

Initialiser une variable globale égale à 0 et qui prend la valeur 1 si la zone geometrique
est au format wkt
En effet le point ramené par l API externe est au format geographique (lattitude, longitude) en wk
il commence par POINT(x, y) et il convient de le mettre dans la projection de la zone géometrique

    class odp extends odp_gen {

        var $wkt=0;


dans la methode setValF, repérer une valeur wkt
        if(substr($val['geom'],0,5)== "POINT"){
```

```
            $this->wkt=1;
            $this->valF['geom'] = null;
        } ...

utiliser les methodes de mise à jour après saisie pour la geometrie :

    function triggermodifierapres($id,&$db,$val,$DEBUG) {
        if($this->wkt==1){
            $this->sig_wkt($id,&$db,$val,$DEBUG);
        }
    }

    function triggerajouterapres($id,&$db,$val,$DEBUG) {
        $id=$this->valF[odp]; // id n est pas valorise en ajout
        if($this->wkt==1){
            $this->sig_wkt($id,&$db,$val,$DEBUG);
        }
    }

    function sig_wkt($id,&$db,$val,$DEBUG){
        // si wkt -> saisie en format binaire wkb pour postgre
        $projection = $db -> getOne("select srid from geometry_columns where f_table_name='".
        $this->table."'");
        $sql ="update ".$this->table." set geom =geometryfromtext('".$val["geom"]."', ".
        $projection." ) where ".$this->table." ='".$id."'";
        $res = $db -> query($sql);
        if (DB :: isError($res)){
            die($res->getMessage()."erreur ".$sql);
        }else{
            $this->msg = $this->msg." "._("le point trouvé par l'API est sauvegardé")." 
            $this->table." ".$id;
        }
    }
```

4.1.7 data_sig

ce chapitre propose de décrire la récupération de données SIG nécessaires à la mise en oeuvre des scripts SIG internes openMairie.

4.1.7.1 Saisir le périmètre de sa commune

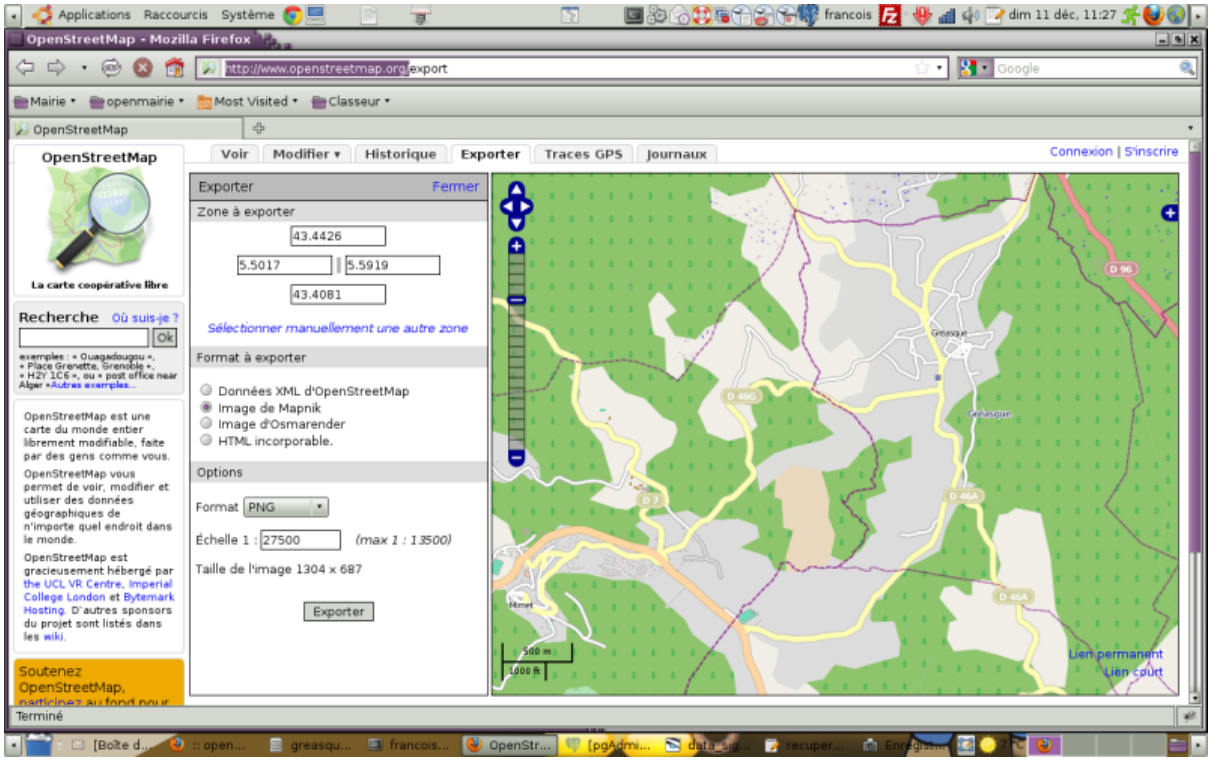
Il s'agit d'adapter ses cartes au périmètre de sa commune

Aller sur openstreetMap http ://www.openstreetmap.org/

Chercher la ville : exemple "Gréasque"

Ajuster la carte aux frontières communales

Aller dans l onglet export et noter les coordonnées géographiques "zone à exporter"



Dans le fichier dyn/var_sig.inc, modifier le tableau de variables avec les coordonnées de la manière suivante

```
$contenu_etendue[0]= array('5.5155,43.4081,5.5781,43.4426');
$contenu_etendue[1]= array('greasque');
```

Modifier les cartes de om_sig_point

4.1.7.2 Récupérer les données de l'IGN

Les bases suivantes sont fournies gratuitement aux collectivités
– base topographique
– base parcellaire
– base adresse
L'application openReferentiel (Vitrolles) permet de créer un référentiel local sur la collectivité

Les données sont fournies par département.

Il faut récupérer les fichiers shape dans une base IGN puis ensuite construire le référentiel de la commune avec openReferentiel.

4.1.7.3 Recupérer des données shape

Il est proposé un exemple de récupération de données "parcelle"

Les données de l'IGN sont fournies aux communes par département.

Insérer le fichier parcelle dans la base (exemple IGN)

```
shp2pgsql -s 2154 -I -D -W LATIN1 PARCELLE.SHP | psql ign
```

Il est créer dans la base "ign" une table parcelle décrite ci dessous

```
gid serial NOT NULL,
        numero character varying(4),
        feuille smallint,
        section character varying(2),
        code_dep character varying(2),
        nom_com character varying(45),
        code_com character varying(3),
        com_abs character varying(3),
        code_arr character varying(3),
        the_geom geometry
```

ainsi qu un enregistrement parcelle dans géometry_columns (postgis est obligatoire dans la base) et un index

Selectionner les parcelles de la commune concernée et inserer les dans une nouvelle table

```
insert into parcelle_greasque       (parcelle, section, commune, geom)
select      section||numero, section, code_dep||code_com, the_geom  from parcelle
    where code_com = '046';
```

Pour mettre à jour le champ surface de parcelle dans openfoncier

```
update parcelle set surface = round(cast(area2d(geom) as numeric), 2)
```

4.1.7.4 Recupération des données de la DGI

Les fichiers textes de la DGI sont dans un format récupéré dans le cadre de l application openCadastre qui reconstitue des tables postgresql.

Les fichiers gémétriques au format EDIGEO ne sont pas récupérés compte tenu de son format non standard et il est préféré utiliser les formats shape de l IGN

L'application openCadastre permet de récupérer les données "texte".

CHAPITRE 5

Historique & Mises à niveau

5.1 Historique & Mises à niveau

5.1.1 La version 4.4

5.1.1.1 Les nouveautés de la version 4.4

...

5.1.1.2 Mettre à niveau depuis openMairie 4.3 vers 4.4

Cette procédure permet de mettre à niveau une application utilisant openMairie version 4.3.0 vers openMairie 4.4.0.

Pour conserver une application fonctionnelle tout au long de la mise à niveau, il est vivement conseillé de :
– suivre les étapes de cette procédure dans l'ordre ;
– ne pas utiliser le générateur lorsque ce n'est pas indiqué.
Consultez la section sur les erreurs connues si des erreurs persistent après la mise à niveau.

Étape 1 : mettre à jour la base de données

5.1.1.2.1.1 La structure La structure de la base de données d'openMairie a changée sensiblement depuis la version 4.3.0. Pour mettre à jour la base de données depuis cette version il faudra exécuter le script SQL `ver_4.4.0.sql`.

Pour PostgresSQL :

```
/data/pgsql/ver_4.4.0.sql
```

Important : Le support de mysql à été abandonné.

Régénérer les fichiers via le générateur.

Vérifier les surcharges des objets : obj/ et sql/

Étape 2 : mise à jour du menu

Suite à l'ajout d'une table contenant les logos et une autre contenant les requêtes, les entrées correspondantes dans le menu devraient êtres ajoutées.

Étape 3 : vérification des requêtes et logos

Vérifier que les logos et requêtes sont dans les bons état/lettres type.

Étape 4 : système de stockage des fichiers

Le système de stockage de fichier ayant été mis à jour, le système utilisé conserve la compatibilité avec les fichiers existant.

Les erreurs connues

5.1.2 La version 4.3

5.1.2.1 Les nouveautés de la version 4.3

...

5.1.2.2 Mettre à niveau depuis openMairie 4.2 vers 4.3

Cette procédure permet de mettre à niveau une application utilisant openMairie version 4.2.0 vers openMairie 4.3.0.

Pour conserver une application fonctionnelle tout au long de la mise à niveau, il est vivement conseillé de :
– suivre les étapes de cette procédure dans l'ordre ;
– ne pas utiliser le générateur lorsque ce n'est pas indiqué.
Consultez la section sur les erreurs connues si des erreurs persistent après la mise à niveau.

Étape 1 : mettre à jour les surcharges du framework

5.1.2.2.1.1 Classe `application` Supprimer l'utilisation de l'attribut :

```
<?php

var $permission_if_right_does_not_exist = true;

?>
```

S'il est utilisé dans une surchage, il doit être remplacée par :

```
<?php

$this->config['permission_if_right_does_not_exist'];

?>
```

Les méthodes surchargés de la classe `om_application` doivent être mises à jour avec leur nouvelle implémentation.

5.1.2.2.1.2 Classe `dbForm` Les méthodes surchargés de la classe `dbForm` doivent être mises à jour avec leur nouvelle implémentation.

5.1.2.2.1.3 Classe `formulaire` Les méthodes surchargés de la classe `formulaire` doivent être mises à jour avec leur nouvelle implémentation.

5.1.2.2.1.4 Classe `table` Les méthodes surchargés de la classe `table` doivent être mises à jour avec leur nouvelle implémentation.

Supprimer l'utilisation de la méthode :

```
<?php

function countHrefColumns($href = array())

?>
```

Si elle est utilisée dans une surchage, elle doit être remplacée par :

```
<?php

function countActions($actions)

?>
```

Étape 2 : mettre à jour la base de données

5.1.2.2.2.1 La structure La structure de la base de données d'openMairie a changée sensiblement depuis la version 4.2.0. Pour mettre à jour la base de données depuis cette version il faudra exécuter le script SQL `ver_4.3.0.sql`.

Pour MySQL :

```
/data/mysql/ver_4.3.0.sql
```

Pour PostgresSQL :

```
/data/pgsql/ver_4.3.0.sql
```

5.1.2.2.2.2 Les tables métier Le générateur gère maintenant plusieurs contraintes :
– `PRIMARY KEY`
– `FOREIGN KEY`
– `UNIQUE`
– `NOT NULL`
En fonction de ces contraintes les fichiers de l'application sont générés différemment par rapport à openMairie version 4.2.0.

5.1.2.2.2.3 `PRIMARY KEY` Ajouter la contrainte SQL PRIMARY KEY.

Le générateur peut maintenant utiliser les clés primaires. Pour créer le champ identifiant, il faudra utiliser la contrainte `PRIMARY KEY` à la place des noms de table en tant que nom de colonne.

5.1.2.2.2.4 `FOREIGN KEY (PostgresSQL)` Ajouter la contrainte SQL FOREING KEY.

Le générateur gère également les clés étrangères des bases PostgresSQL. Pour créer des références, il faudra utiliser la contrainte `FOREIGN KEY` à la place des noms de table étrangères en tant que nom de colonne.

5.1.2.2.2.5 `UNIQUE`
– **Ajouter la contrainte SQL UNIQUE.**
– **Mettre a jour les fichiers de surcharge du répertoire obj/.**
La contrainte `UNIQUE` permet maintenant de gérer automatiquement les champs uniques. Il n'est plus nécessaire de surcharger la méthode `verifier` des modèles pour gérer ce type de champ. Il faudra nettoyer les surcharges de `verifier` en supprimant la vérification manuelle des champs requis et les remplacer par des contraintes `UNIQUE` dans la base de données.

5.1.2.2.2.6 NOT NULL

– **Ajouter la contrainte NOT NULL aux champs requis.**
– **Supprimer la clause DEFAULT des champs requis.**
– **Supprimer la contrainte NOT NULL des champs non-requis ou ajouter la clause DEFAULT en fonction du besoin.**
– **Mettre a jour les fichiers de surcharge du répertoire obj/.**
– **Générer.**

Toutes les colonnes `NOT NULL` généreront des champs requis. Des champs qui n'étaient pas requis dans la version 4.2.0 peuvent donc l'être dans la version 4.3.0 après une génération. Il faut donc supprimer la contrainte `NOT NULL` des colonnes qui ne sont pas réellement requises par l'application ou ajouter une valeur par defaut avec la clause `DEFAULT`.

Concernant les champs requis par l'application. Il n'est plus nécessaire de surcharger la méthode `verifier` des modèles pour gérer ce type de champ. Il faudra nettoyer les surcharges de `verifier` en supprimant la vérification manuelle des champs requis et les remplacer par des contraintes `NOT NULL` sans clause `DEFAULT` dans la base de données.

Important : Vous pouvez générer à nouveau l'application à partir d'ici.

Étape 3 : mettre à jour les fichiers de surcharge du répertoire `sql/`

5.1.2.2.3.1 Alias des tables étrangères Prefixer le nom des colonnes étrangères par l'alias généré dans gen/sql/.

Le générateur peut donner à une table étrangère un alias unique. Cela permet d'effectuer plusieurs jointures sur une même table sans avoir d'erreur d'ambiguïté avec les nom des colonnes. Pour cela, dans les fichiers du répertoire `sql/` contenant plusieurs référence vers une même table étrangère, les noms des colonnes provenant de ces tables étrangères devront être préfixés par l'alias adéquat. Cet alias apparaît dans la définition de la variable `$table` dans les fichiers générés du répertoire `gen/sql/`.

5.1.2.2.3.2 La clause ORDER BY Supprimer les surcharges de la variable $tri = 'ORDER BY libelle', ce tri est généré par défaut.

Le générateur crée maintenant une clause SQL `ORDER BY` pour chaque modèle. Le tri par défaut se fait sur une éventuelle colonne `libelle`. Si elle n'existe pas la deuxième colonne de la table est utilisée, sinon la clé primaire.

Note : Dans le cas où la deuxième colonne d'une table est utilisée comme libellé, si cette colonne est une clé étrangère, alors le tri se fera sur le libellé de la table étrangère.

5.1.2.2.3.3 Les actions du tableau

5.1.2.2.3.4 Les actions d'openMairie

– **Remplacer $href[0] par $tab_actions['corner']['ajouter'].**
– **Remplacer $href[1] par $tab_actions['left']['modifier'].**
– **Remplacer $href[2] par $tab_actions['left']['supprimer'].**

Pour surcharger l'action ajouter, il faut maintenant surcharger `$tab_actions['corner']['ajouter']` et non plus `$href[0]` :

```
<?php

$tab_actions['corner']['ajouter'] =
    array('lien' => 'form.php?obj='.$obj.'&amp;action=0',
          'id' => '&amp;advs_id='.$advs_id.'&amp;tricol='.$tricol.'&amp;valide='.$valide.'&amp;ret
          'lib' => '<span class="om-icon om-icon-16 om-icon-fix add-16" title="'._('Ajouter').'">
          'rights' => array('list' => array($obj, $obj.'_ajouter'), 'operator' => 'OR'),
```

```
        'ordre' => 10,);

?>
```

Pour surcharger l'action modifier, il faut maintenant surcharger `$tab_actions['left']['modifier']` et non plus `$href[1]` :

```
<?php

$tab_actions['left']['modifier'] =
    array('lien' => 'form.php?obj='.$obj.'&amp;action=1'.'&amp;idx=',
          'id' => '&amp;premier='.$premier.'&amp;advs_id='.$advs_id.'&amp;recherche='.$recherche1
          'lib' => '<span class="om-icon om-icon-16 om-icon-fix edit-16" title="'._('Modifier').'
          'rights' => array('list' => array($obj, $obj.'_modifier'), 'operator' => 'OR'),
          'ordre' => 20,);

?>
```

Pour surcharger l'action de contenu, il faut maintenant surcharger `$tab_actions['content']` et non plus `$href[1]` :

```
<?php

 $tab_actions['content'] = $tab_actions['left']['modifier'];

?>
```

Pour surcharger l'action supprimer, il faut maintenant surcharger `$tab_actions['left']['supprimer']` et non plus `$href[2]` :

```
<?php

$tab_actions['left']['supprimer'] =
    array('lien' => 'form.php?obj='.$obj.'&amp;action=2&amp;idx=',
          'id' => '&amp;premier='.$premier.'&amp;advs_id='.$advs_id.'&amp;recherche='.$recherche1
          'lib' => '<span class="om-icon om-icon-16 om-icon-fix delete-16" title="'.__('Supprimer'
          'rights' => array('list' => array($obj, $obj.'_supprimer'), 'operator' => 'OR'),
          'ordre' => 30,);

?>
```

5.1.2.2.3.5 Les actions personnalisées Redéfinir les actions avec la nouvelle manière.

Les actions personnalisées doivent être défini selon la nouvelle manière. Exemple :

```
<?php

$tab_actions['left']['edition'] = array(
    'lien' => '../pdf/pdfetat.php?obj=om_collectivite&amp;idx=',
    'id' => '',
    'lib' => '<span class="om-icon om-icon-16 om-icon-fix pdf-16" title="'._('Edition').'">'._('E
    'ajax' => false,
    'ordre' => 21,
);

?>
```

5.1.2.2.3.6 Définition de l'action

La première clé de `$tab_actions` permet choisir la position d'affichage :
- `corner` pour les actions en coin ;
- `left` pour les actions de gauche.

La seconde clé de `$tab_actions` permet de définir la nouvelle action. Cette clé doit être différente de : `ajouter`, `consulter`, `modifier` et `supprimer`.

Les clés `lien`, `id` et `lib` s'utilise de la même manière qu'avant.

5.1.2.2.3.7 Définition du mode d'affichage en sous-tableau La clé `ajax` permet d'indiquer si l'action doit être affichée en ajax ou non dans les sous-tableaux :
– `true`, l'action utilisera la fonction `ajaxIt()` ;
– `false`, l'action n'utilisera pas la fonction `ajaxIt()`.

5.1.2.2.3.8 Définition de l'ordre d'affichage La clé `ordre` permet de déterminer l'odre d'affichage par rapport aux autres actions.

Chaque action dispose d'une valeur numérique permettant de définir sa place au sein d'une position. L'action numéro 1 s'affichera en premier, l'action numéro 10 s'affichera après les actions de numéro inférieur, etc.

Ordre des actions par défaut d'openMairie :
– ajouter à pour ordre 10 dans la position `corner` ;
– consulter à pour ordre 10 dans la position `left`.
Si la position `corner` est sélectionnée :
– 9, l'action s'affichera avant l'action `ajouter` ;
– 11, l'action s'affichera après l'action `ajouter`.
Si la position `left` est sélectionnée :
– 9, l'action s'affichera avant l'action `consulter` ;
– 11, l'action s'affichera après l'action `consulter`.

5.1.2.2.3.9 Définition des droits d'affichage La clé `rights` permet de définir le ou les droits nécessaire à l'utilisateur pour visualiser cette action. Cette clé est optionnelle. Si `rights` n'existe pas, tous les utilisateurs pourront visualiser cette action s'ils peuvent visualiser le tableau correspondant.

La clé `list` permet de définir le tableau des droits nécessaire.

La clé `operator` permet de définir l'opérateur utilisé pour pour vérifier les droits de la liste `list` :
– `OR`, l'utilisateur doit avoir au moins un droit ;
– `AND`, l'utilisateur doit avoir tous les droits.

Les erreurs connues

5.1.3 La version 4.2

5.1.3.1 Les nouveautés de la version 4.2

...

5.1.3.2 Mettre à niveau depuis openMairie 4.1 vers 4.2

La version 4.2.0 du framework prend en charge plus de fonctionnalités et donne toutes possibilité de surcharge aux applications :
– surcharge des objets génères par le generateur ;
– surcharge des composants de base openMairie stocké dans core ;
– surcharge de la présentation de base (dans img et css), des thèmes (om-theme) dans app/css app/img ;
– surcharge du javascript de base app/js/script.js.

EXTERNALS.txt

Vider les 9 repertoires concernés avant de lancer externals.

Appliquer le fichier EXTERNALS.txt d'openmairie :

```
core  svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/core/
spg   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/spg/
scr   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/scr/
lib   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/lib/
css   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/css/
js    svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/js/
img   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/img/
pdf   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/pdf/
php   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/php/

om-theme svn://scm.adullact.net/svnroot/openmairie/externals/om-theme/kinosura/tags/1.0.0
```

Deux répertoires sont remarquables :
– core contient maintenant la librairie openMairie (4.2.0) ;
– om_theme est le theme de l'application.
Les dossiers suivant sont spécifiques à l'application elle même :
– app, scripts spécifiques de l'application, noter dans specific.txt les spécificités ;
– data, scripts SQL d'initialisation de la base de données ;
– dyn, paramétrage de l application ;
– gen, fichiers générés de l'application ;
– obj, surcharge des objets générés dans gen/obj/, surcharge du core d'openMairie om_dbformdyn, om_formulaire (4.2.0), om_application (util) ;
– sql, requêtes SQL surchargeant les fichiers de gen/sql ;
– tmp, fichiers temporaires de l'application (ajouter les droits d'écriture pour le serveur web) ;
– trs, fichiers uploadés par l'application (ajouter les droits d'écriture pour le serveur web.

Regenerer les tables avec genfull.php

Ajouter les droits d'écriture à www-data dans gen :

```
$ sudo chmod -R 777 gen
```

Modifier les paramètres dyn

locales.inc.php (charset) include.inc.php (core)

Dans obj/

Ajouter om_table.class.php, om_dbform.class.php, et om_formulaire.class.php.

Evolution om_sig_point vers om_sig_map

om_sig_map est le nouvel outil SIG d'openMairie.

Ne concerne que PostgresSQL.

Executer le script data/pgsql/ver4.2.0.sql.

Regénérer les 4 nouvelles tables. Ajouter les scripts spécifiques dans /obj et /sql/pgsql.

CHAPITRE 6

Règles

6.1 Règles

6.1.1 Convention de codage

La convention de codage openMairie s'applique à tout le code qui fait partie de la distribution officielle d'openMairie ainsi qu'aux applicatifs de la gamme. La convention de codage permet de conserver un code consistant et de le rendre lisible et maintenable facilement par les développeurs openMairie.

6.1.1.1 L'indentation du code

Pour améliorer la lisibilité, il faut utiliser une indentation de 4 espaces et non pas des tabulations. En effet, les éditeurs de texte interprètent différemment les tabulations alors que les espaces sont tous interprétés de la même façon. De plus lors de commit, les historiques des gestionnaires de versions (CVS ou SVN) sont faussés par ces caractères.

Il est recommandé que la longueur des lignes ne dépasse pas 75 à 85 caractères.

6.1.1.2 Encodage des fichiers

L'encodage des fichiers doit être UTF-8.

6.1.1.3 Tags dans le code PHP

Il faut utiliser toujours < ?php ?> pour délimiter du code PHP, et non la version abrégée < ? ?>. Cela est la méthode la plus portable pour inclure du code PHP sur des systèmes d'exploitations disposant de configurations différentes.

6.1.1.4 HTML Valide et W3C

Le Code HTML rendu doit être validé et correspondre aux standards W3C.

6.1.1.5 Les commentaires dans le code

Tous les fichiers PHP doivent avoir un entête de ce style

```
<?php
/**
 * Courte description du fichier
 *
 * Description plus détaillée du fichier (si besoin en est)...
 *
```

```
 * @package openmairie
 * @version SVN : $Id$
 */

?>
```

6.1.1.6 Images

Les fichiers images ajoutés dans les applications openMairie doivent être au format PNG (Portable Network Graphics). Ce format permet d'obtenir des images de qualité avec des propriétés de transparence.

6.1.2 Versionnage

6.1.2.1 Convention de numérotation des versions

Cette convention concerne le framework et les applications. Il est convenu de numéroter les versions sur 3 chiffres séparés par des points.

exemple : openMairie 4.0.0

Le premier chiffre représente une version majeure

Le deuxième chiffre est une évolution mineure

Le troisième chiffre est une correction de bug

Les versions beta sont indiqués en fin de numérotation et ne sont jamais maintenues

    openmairie_exemple_4.0.0beta

Seule la dernière version opérationnelle est maintenu

Exemple de versionning (complément de la réunion "dev-openMairie" du 13 juin 2012) :

```
4.2.0alpha1     première version non testée
4.2.0beta1      première version testée par le développeur
4.2.0rc1        première version testée en production (1 site)
4.2.0           première version stable généralisable
```

6.1.3 Documentation

Ce paragraphe vise à regrouper les bonnes pratiques qui permettent à toute la communauté openMairie de travailler sur les mêmes bases et avec les mêmes références concernant les outils de rédaction et de publication de documentation afin de faciliter la contribution et d'obtenir un rendu homogène.

Voici les postulats :
– chaque logiciel/application possède une seule documentation Sphinx qui contient un manuel d'utilisation et un guide technique biens séparés
– le code source de la documentation doit être déposé dans un dépôt GIT sur github.com
– la génération de la documentation est gérée de manière automatique par readthedocs.org
– docs.openmairie.org est l'unique interface d'accès pour toutes les documentations
– un lien dans le footer de chaque application permet d'accéder à la version de l'application du manuel d'utilisateur sur le site docs.openmairie.org

CHAPITRE 7

Outils

7.1 Outils

Cette section vise à rassembler des liens, des informations, des tutoriels sur des outils qui ne font pas partie du Framework mais qui sont utilisés par la communauté :

7.1.1 Apache Subversion (SVN)

Site officiel du projet SVN

7.1.1.1 Pré-requis

Installer subversion : http ://subversion.apache.org/packages.html

Il existe également des outils graphiques comme TortoiseSVN (Windows).

7.1.1.2 L'arborescence

Voici l'arborescence standard d'un projet versionné sur un SVN :

```
/trunk/
/tags/
/branches/
```

- trunk/ : la version en cours de développement.
- tags/ : les différentes versions publiées. Les dossiers dans tags/ sont des copies du dossier trunk/ a un instant précis. Ils permettent de fixer une version pour la publier. Il est interdit d'effectuer une modification dans un de ces dossiers, la bonne méthode étant de faire la modification dans le trunk/ et de faire une nouvelle version dans le dossier tags/.
- branches/ : ...

7.1.1.3 Les règles d'or

- Ne jamais commiter dans un tag.
- Ne jamais commiter sans message de commit.
- Ne jamais tagger une version qui contient des externals vers un 'trunk'.

7.1.1.4 Les commandes basiques à connaître

Récupérer une copie locale :

```
svn co svn+ssh://nom-du-développeur@scm.adullact.net/openmairie/openmairie_exemple/trunk
    openmairie_exemple
```

Mettre à jour sa copie locale :

```
svn up
```

Voir l'état de sa copie locale :

```
svn st
```

Voir la différence entre sa copie locale et le dépôt :

```
svn diff
```

```
svn ci
```

7.1.1.5 Externals

C'est une propriété sur le dépôt SVN permettant d'importer du code provenant d'un dépôt différent.

Le fichier EXTERNALS.txt :

```
#
# created by: svn propset svn:externals -F ./EXTERNALS.txt .
#

core  svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/core/
spg   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/spg/
scr   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/scr/
lib   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/lib/
css   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/css/
js    svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/js/
img   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/img/
pdf   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/pdf/
php   svn://scm.adullact.net/svnroot/openmairie/openmairie_exemple/tags/4.2.0/php/

om-theme svn://scm.adullact.net/svnroot/openmairie/externals/om-theme/kinosura/tags/1.0.0
```

Appliquer les propriétés externals :

```
svn propset svn:externals -F ./EXTERNALS.txt .
svn up
svn ci
```

attention le repertoire ne doit pas etre existant (copie de travail verouillée)

7.1.1.6 Keywords

7.1.1.7 Les clients graphiques

Il est recommandé de savoir utiliser et d'utiliser subversion en ligne de commande mais il existe quelques clients graphiques qui permettent de réaliser certaines opérations d'une manière plus conviviale.
– Meld
– TortoiseSVN
– ...

7.1.1.8 Tutoriaux

Importer un nouveau projet

Un nouveau projet est une nouvelle application qui se base sur la dernière version taggée d'openmairie_exemple. Ce tutorial contient certains pré-requis comme la création du projet sur la forge, le fait d'avoir un utilisateur avec les droits corrects sur le projet, le fait d'avoir consulter la dernière version taggée d'openmairie_exemple

On se positionne dans la dossier tmp pour récupérer la dernière version d'openmairie_exemple

```
cd /tmp
svn export --ignore-externals svn://scm.adullact.net/svnroot/openmairie/
    openmairie_exemple/tags/<DERNIERE_VERSION_OPENMAIRIE_EXEMPLE>/ openexemple
```

On cré l'arborescence standard sur le dépôt

```
svn mkdir svn+ssh://<NOM_DU_DEVELOPPEUR>@scm.adullact.net/scmrepos/svn/<NOUVEAU_PROJET>/trunk
svn mkdir svn+ssh://<NOM_DU_DEVELOPPEUR>@scm.adullact.net/scmrepos/svn/<NOUVEAU_PROJET>/tags
svn mkdir svn+ssh://<NOM_DU_DEVELOPPEUR>@scm.adullact.net/scmrepos/svn/<NOUVEAU_PROJET>/branches
```

On se positionne dans le dossier précédemment importé pour importer sur le dépôt son contenu

```
cd openexemple
svn import . svn+ssh://<NOM_DU_DEVELOPPEUR>@scm.adullact.net/svnroot/<NOUVEAU_PROJET>/trunk
```

On se positionne dans son dossier de développement pour créer la copie locale du projet

```
cd ~/public_html/
svn co svn+ssh://<NOM_DU_DEVELOPPEUR>@scm.adullact.net/scmrepos/svn/<NOUVEAU_PROJET>/trunk
        <NOUVEAU_PROJET>
```

On se positionne dans le dossier php de l'application pour appliquer les externals

```
cd <NOUVEAU_PROJET>/php
svn propset svn:externals -F ./EXTERNALS.txt .
svn up
svn ci
```

Publier une nouvelle version

Ce tutorial contient certains pré-requis comme le fait d'avoir un utilisateur avec les droits corrects sur le projet ou connaître comment incrémenter le numéro de version de l'application à publier.

Avant de publier une application, il faut vérifier que l'EXTERNALS de la librairie openMairie ne pointe pas vers le 'trunk'. Pour cela

```
less php/EXTERNALS.txt

#
# created by: svn propset svn:externals -F ./EXTERNALS.txt .
#

openmairie svn://scm.adullact.net/svnroot/openmairie/openmairie/trunk/
fpdf svn://scm.adullact.net/svnroot/openmairie/externals/fpdf/tags/1.6-min/
pear http://svn.php.net/repository/pear/pear-core/tags/PEAR-1.9.1/
db http://svn.php.net/repository/pear/packages/DB/tags/RELEASE_1_7_13/
```

Ici on voit que openmairie pointe vers le 'trunk'. Nous devons d'abord publier la librairie

```
svn cp svn+ssh://<NOM_DU_DEVELOPPEUR>@scm.adullact.net/openmairie/openmairie/trunk
       svn+ssh://<NOM_DU_DEVELOPPEUR>@scm.adullact.net/openmairie/openmairie/tags/<NOUVELLE_VERSIO
```

Le message pourra être : Tag openmairie <NOUVELLE_VERSION>.

Ensuite il faut changer les EXTERNALS.txt. On remplace dans le fichier php/EXTERNALS.txt, le trunk par la nouvelle version

```
vim php/EXTERNALS.txt

#
# created by: svn propset svn:externals -F ./EXTERNALS.txt .
#

openmairie svn://scm.adullact.net/svnroot/openmairie/openmairie/tags/<NOUVELLE_VERSION>/
fpdf svn://scm.adullact.net/svnroot/openmairie/externals/fpdf/tags/1.6-min/
pear http://svn.php.net/repository/pear/pear-core/tags/PEAR-1.9.1/
db http://svn.php.net/repository/pear/packages/DB/tags/RELEASE_1_7_13/
```

Ensuite on applique le nouveau propset externals une fois placé dans le dossier php (Attention de ne pas oublier le "." dans la commande svn propset)

```
cd php/
svn propset svn:externals -F ./EXTERNALS.txt .
svn up
```

Ici en faisant un svn info sur le dossier openmairie, nous devons obtenir une URL comme ceci

```
svn info openmairie/
URL : svn://scm.adullact.net/svnroot/openmairie/openmairie/tags/<NOUVELLE_VERSION>
```

Si tout est ok nous pouvons valider nos modifications puis passer à la publication de l'application

```
svn ci
```

Ici on fait une copie du 'trunk' vers le dossier 'tags' de l'application openmairie_exemple

```
svn cp svn+ssh://<NOM_DU_DEVELOPPEUR>@scm.adullact.net/openmairie/openmairie_exemple/trunk
       svn+ssh://<NOM_DU_DEVELOPPEUR>@scm.adullact.net/openmairie/openmairie_exemple/tags/<NOUVEL
```

svn utilisation

il est propose dans ce chapitre de lister quelques commandes utiles en cas de conflit testées en svn

type de fichier

```
A Ajout de nouveaux éléments à la version locale
M éléments modifiés localement (par rapport à la version de SVN) ;
? éléments inconnus de SVN (non présents dans la version de SVN) ;
U pour les éléments modifiés dans SVN (par rapport à la version locale) ;
C pour les éléments différents entre les versions locale et SVN, et qui posent un conflit à règle
?D fichier supprimés (a verifier)
```

revert et diff :

```
svn revert nomfichier                        // remet dans le dernier etat du svn (soit pas del so
svn diff nomdossier ou nomfichier   // affiche les modifications r/r au dernier svn up
```

resolution de conflit

```
svn st
!     C openmairie_exemple/trunk/authors.txt
      >   local édition, suppression entrante sur mis à jour

svn revert openmairie_exemple/trunk/authors.txt
    'openmairie_exemple/trunk/authors.txt' réinitialisé
```

deplacer un dossier sur le svn -> commande mv

```
Exemple : on a créé trunk/trunk/dossiers_source

D'abord on renomme le premier dossier trunk en dossier branches
> svn mv svn+ssh://fraynaud@scm.adullact.net/scmrepos/svn/openboisson/trunk
    svn+ssh://fraynaud@scm.adullact.net/scmrepos/svn/openboisson/branches
cela fait branches/trunk/dossiers_souce

Ensuite on déplace le dossier trunk qui se trouve maintenant dans branches à la racine du dépôt
> svn mv svn+ssh://fraynaud@scm.adullact.net/scmrepos/svn/openboisson/branches/trunk
    svn+ssh://fraynaud@scm.adullact.net/scmrepos/svn/openboisson/trunk


cela fait trunk/dossiers_source
```

creer - deplacer (autre exemple) - detruire un repertoire sur svn

```
creer un dossier documentation sur svn depuis une copie loacle
svn import documentation svn+ssh://fraynaud@scm.adullact.net/scmrepos/svn/opencimetiere/documentat

renomer = renommer sur le svn trunk en temp
svn rename svn+ssh://fraynaud@scm.adullact.net/scmrepos/svn/opencimetiere/documentation/trunk
           svn+ssh://fraynaud@scm.adullact.net/scmrepos/svn/opencimetiere/documentation/temp

move = deplacer le dossier temp/trunk vers trunk
svn mv svn+ssh://fraynaud@scm.adullact.net/scmrepos/svn/opencimetiere/documentation/temp/trunk
       svn+ssh://fraynaud@scm.adullact.net/scmrepos/svn/opencimetiere/documentation/trunk

delete  detruire le repertoire temp
svn del svn+ssh://fraynaud@scm.adullact.net/scmrepos/svn/opencimetiere/documentation/temp
```

Creation d une nouvelle version

```
Copie en tag de la version

svn cp  svn+ssh://fraynaud@scm.adullact.net/scmrepos/svn/openboisson/trunk
        svn+ssh://fraynaud@scm.adullact.net/scmrepos/svn/openboisson/tags/1.0.0beta

export dans un repertoire local openmairie_debitboisson_1.0.0beta sans les repertoires .svn

svn export  svn+ssh://fraynaud@scm.adullact.net/scmrepos/svn/openboisson/tags/1.0.0beta
        openmairie_debitboisson_1.0.0beta
```

7.1.2 Concurrent versions system (CVS)

Site officiel du projet CVS

Il est noté dans ce chapitre les commandes de bases de cvs à titre d'indicatif. Les développeurs d openMairie souhaitent s'orienter sur SVN qui est plus facile dans la gestion des répertoires (notament la suppression) et dans la mise en place de composants externes (externals)

7.1.2.1 forge > local

update

```
$ cvs up
$ cvs up -A // depuis la branche principale
$ cvs up -r branch_name // depuis une branche
$ cvs up -r tag_name  // depuis un tag
```

```
$ cvs up -d  //creer sur le poste local les nouveaux repertoires
$ cvs up -C  //Annulation des modifications locales -> les fichiers remplacés sont sauvés en .#fi
```

message

```
A Ajout de nouveaux éléments à la version locale
M éléments modifiés localement (par rapport à la version de CVS) ;
? éléments inconnus de CVS (non présents dans la version de CVS) ;
U pour les éléments modifiés dans CVS (par rapport à la version locale) ;
C pour les éléments différents entre les versions locale et CVS, et qui posent un conflit à règle
D fichier supprimés
```

7.1.2.2 local -> forge

commit

```
$ cd <répertoire désiré>
$ cvs commit [-m "message explicite de la modification"] [<arborescence>]
$ cvs ci -m "ajout des fichiers" // avec message
$ cvs ci module_name // dans la branche principale
$ cvs ci -d branch_name module_name // dans une branche
$ cvs ci -r tag module_name // depuis un tag
$ cvs ci -r 3.0 //changer la révision (doit être plus grand que tous les numéros existants)
```

Add

```
$ cd <répertoire désiré>
$ cvs add [-m "message explicite de l'ajout"] <arborescence>
$ cvs add file1 file2 ...
cvs add /data/pgsql*  // * tous les fichiers du repertoire  ...
L'ajout d'un fichier n'est possible que depuis le répertoire le contenant ;
Les fichiers d'un répertoire non ajouté ne sont pas visibles par CVS (lors de l'affichage des dif
L'ajout d'éléments est effectif dans CVSROOT uniquement après la mise à jour de la version de CVS
```

rm

```
Pour supprimer des éléments à la version locale, afin de les supprimer ensuite définitivement de
$ cd <répertoire désiré>
$ cvs remove [-f] <arborescence>
 * -f : pour effacer le fichier avant de le supprimer.

Il faut d'abord supprimer le fichier du répertoire
$ rm file1 file2 ...
$ cvs rm file1 file2 ...
$ cvs ci -m "suppression des fichiers
Supprimer un répertoire n est pas possible directement, il faut aller le supprimer dans le serveu
```

7.1.2.3 local

Suppression d'une copie locale

```
// pour éviter d'effacer un checkout en oubliant nos modifications
Pour effacer les sources présentes sur le poste local et faire confiance au repository :
$ cd <répertoire désiré>
$ cvs release [-d] <arborescence>
* -d : pour effacer définitivement l'arborescence.
$ cvs release -d module
```

status

```
Pour obtenir le détail (statut) de la version locale d'une arborescence :
$ [ cd <répertoire désiré> ]
$ cvs status <arborescence>
cvs st -> status
```

diff/log

```
cvs diff
cvs log <nomfichier>
cvs diff <nomfichier>     [local /differentiel]
cvs diff -r 1.5 -r 1.6 nomfichier [entre 2 versions]
```

7.1.2.4 export

exemple opencimetiere_facture
– se mettre dans le repertoire : openmairie_cimetiere_facture
– taguer la version

```
$ cvs tag version_2_03 (commence par une lettre / pas de point)
```

– faire l export

```
$ cvs export - r version_2_03 -d version/opencimetiere_facture_2.03 openmairie_cimetiere_facture
                            [      version   ]   [repertoire export ] [module]
```

7.1.2.5 tag

Les identifiants logiques (noms donnés à une version par un utilisateur) sont différents des identifiants CVS (du type 1.1.2.1). La gestion d'identifiant (ou tag) d'une arborescence se fait ainsi

```
$ [ cd <répertoire désiré> ]
$ cvs tag [-R] [-d -r] <nom de l'identifiant> [<arborescence>]
$ cvs tag tag_name // creer un tag
* -R : commande appliquée récursivement sur les sous-répertoires ;
* -d -r : suppression de l'identifiant existant.
```

Exportation (mêmes options que cvs check out)

Pour exporter les sources du projet en vue d'une livraison (pas de répertoires CVS dans l'arborescence)

```
$ cd <répertoire désiré>
$ cvs export (-r <nom du tag> | -D <date désirée>) <arborescence>

$ cvs export
```

Les fichiers .cvsignore sont exportés et apparaissent dans l'arborescence contrairement aux répertoires CVS ; Des problèmes apparaissent lors d'export de fichiers binaires sur plateformes hétérogènes. Par exemple, un export sur PC transforme les retours chariots (n -> r n)

7.1.2.6 Import

requete cvs

```
cvs -d :pserver:user@cvs.mpl.ird.fr:/projet login
       (1)     (2)      (3)          (4)
        |       |        |            |
        |       |        |            +- Répertoire du
        |       |        |    SERVEUR contenant les sources
        |       |        |              (racine)
        |       |        |
```

```
|       |        |
|       |        +-------- adresse du SERVEUR CVS
|       |
|       +------ Votre login à vous sur le SERVEUR
|
+----- le type d'authentification
```

7.1.2.7 checkout

Pour récupérer les sources du projet en local

```
$ cd <répertoire désiré>
$ cvs checkout <arborescence>
```

7.1.2.8 Divers

aide

```
$ cvs -H nomdecommande
```

log des commit : Pour obtenir l'historique d'une arborescence

```
$ [ cd <répertoire désiré> ]
$ cvs log <arborescence>
```

L'historique affiche les différents identifiants (ou tag) et les différentes versions de l'arborescence concernée sous CVS ;

L'historique sur un répertoire affiche récursivement les historiques des fichiers le composant.

.cvsignore

La présence de fichier(s) .cvsignore dans un répertoire permet de dire à CVS d'ignorer certains types de fichiers

```
$ cd <répertoire désiré>
$ cat .cvsignore
*.jpg *.htm
```

7.1.2.9 Changer le système de gestion des version de CVS vers SVN sur la forge de l'adullact

Le but de ce tutorial est de changer le système de gestion de version sur un projet existant sur la forge de l'adullact.

Pré-requis

- Le projet sur l'adullact en CVS <NOM_DU_PROJET>
- Le nom du module à récupérer <NOM_DU_MODULE>
- Les droits d'administration sur le projet <NOM_DU_DEVELOPPEUR>

Etape 1 - Récupérer le code du CVS

```
cvs -d :pserver:anonymous@scm.adullact.net:/cvsroot/<NOM_DU_PROJET> login
cvs -d :pserver:anonymous@scm.adullact.net:/cvsroot/<NOM_DU_PROJET> export -DNOW <NOM_DU_MODULE>
```

Important : si un mot de passe est demandé, un mot de passe vide fera l'affaire.

A cette étape, il est recommandé de faire une archive du dossier '<NOM_DU_MODULE>' qui vient d'être exporté pour le sauvegarder.

Etape 2 - Changer le type de dépôt

En tant qu'administrateur, aller dans l'onglet 'Sources' puis cliquer sur le lien 'Administration'. Choisir alors SVN puis cliquer sur le bouton 'Mettre à jour'.

Il faut ensuite attendre (le temps d'attente est variable entre 30 minutes et plusieurs heures) que le dépôt subversion soit activé.

Etape 3 - Créer la structure du dépôt

Ici nous allons créer la structure standard d'un dépôt Subversion :
– trunk
– tags
– branches

```
svn mkdir svn+ssh://<NOM_DU_DEVELOPPEUR>@scm.adullact.net/svnroot/<NOM_DU_PROJET>/trunk svn+ssh:/
```

Etape 4 - Importer le code sur le nouveau dépôt Subversion

7.1.2.9.5.1 Cas 1 Si l'application doit être utilisée telle qu'elle a été récupérée sur le CVS, alors nous allons l'importer directement dans le dossier 'trunk'.

```
svn import <NOM_DU_MODULE> svn+ssh://<NOM_DU_DEVELOPPEUR>@scm.adullact.net/svnroot/<NOM_DU_PROJET
```

7.1.2.9.5.2 Cas 2 Dans le cas de figure où l'application va être migrée vers OM4, nous allons placer le code récupéré sur le CVS dans une branche du dépôt correspondant à sa version.

```
svn import <NOM_DU_MODULE> svn+ssh://<NOM_DU_DEVELOPPEUR>@scm.adullact.net/svnroot/<NOM_DU_PROJET
```

Dans ce cas, il faut donc importer le nouveau code dans le dossier 'trunk' :
– Soit le développement du projet n'a pas encore commencé et il suffit de suivre le tutorial "versions/svn/Importer un nouveau projet".
– Soit le développement du projet a déjà commencé et il suffit d'importer le dossier en cours de développement (attention : il ne faut pas que des dossiers .svn soient présents dans ce dossier et il faut prendre soin de supprimer les dossiers récupérés depuis les EXTERNALS avant l'import) :

  ```
  svn import <NOM_DU_DOSSIER> svn+ssh://<NOM_DU_DEVELOPPEUR>@scm.adullact.net/svnroot/<NOM_DU_PRO
  ```

  Il faut bien sur valider les EXTERNALS pour récupérer les librairies externes.

7.1.3 GIT

Site officiel du projet GIT

7.1.4 Meld

Site officiel du projet Meld

exemple d utilisation :

meld openmairie_recensement/ svn.openmairtrunck/trunk

7.1.5 POEdit

POEdit est un éditeur de traductions de chaînes en plusieurs langues. Les chaînes présentes dans l'interface des applications openMairie sont celles présentes dans le code. Elles ne peuvent pas comporter d'accent.

Les étapes sont les suivantes :

1 - avoir dans le code .php les chaînes à traduire, en texte sans accent ni caractères spéciaux

2 - préparer les dossiers de traductions dans le dossier de locales

3 - avoir préalablement installé et configuré POEdit

4 - configurer le projet dans POEdit et effectuer scan du code afin de détecter les chaînes à traduire

5 - traduire les chaînes dans POEdit et sauvegarder

7.1.5.1 Spécification dans le code des chaînes à traduire

Les chaînes peuvent être traduites, soit en français accentué, soit dans d'autres langues. Pour cela il est nécessaires qu'elles soient présentes dans les fichiers.php en respectant la syntaxe suivante

```
``_('Ma chaine a traduire, sans accent')``
```

Toutes les chaînes de caractères correspondant aux noms de tables et de champs sont générées par le générateur et sont ainsi directement disponibles.

7.1.5.2 Préparation des dossiers de locales

Chaque application openMairie comporte, à la racine, un dossier appelé `locales`. Ce dossier comporte une structure de type

```
locales/
    fr_FR/
        LC_MESSAGES/
            openmairie.po
            openmairie.mo
```

Il est nécessaire de créer un dossier de langue (par exemple `en_US`) avec son sous-dossier `LC_MESSAGES` pour chaque langue supplémentaire.

Le fichier .po contient les définitions de traductions ; il peut être modifié au moyen de POEdit ou directement depuis un éditeur de texte simple.

Le fichier .mo contient les traductions sous une forme compilée. Il est généré par POEdit automatiquement lors de chaque sauvegarde des traductions.

7.1.5.3 Installation et configuration de POEdit

Installation

POEdit est disponible nativement dans la plupart des systèmes linux. Il est possible de le télécharger depuis le site officiel pour tous Linux, Windows et Mac OSX.

http ://www.poedit.net/download.php

Sous Linux Ubuntu ou Debian, il faut, en root, exécuter la commande `apt-get install poedit`

Gestion de plusieurs langues

Une fois installé, il faut s'assurer que les `locales` (fichiers de définition de langues) sont correctement installés sur le système.

Sous Linux Ubuntu, pour ajouter une locale, il faut ajouter sa définition dans le fichier de pays correspondant (exemple : `/var/lib/locales/supported.d/fr`) puis lancer la commande, en root, `dpkg-reconfigure locales`. Cela ne sera nécessaire que pour ajouter la prise en charge d'une nouvelle langue.

7.1.5.4 Configuration d'un projet dans POEdit

- Cliquer sur le bouton 'Créer un nouveau projet de traduction', même si le projet comporte déjà une traduction,
- Saisir le nom du projet et le chemin complet du dossier contenant le projet (chemin complet permettant d'aller à la racine du logiciel openMairie)
- Valider. Cela crée le projet. Les fichiers .po existants sont automatiquement détectés.
- Cliquer alors sur le bouton 'Mettre à jour tous les catalogues du projet'. Un scan de code est alors effectué par POEdit. Cela va parcourir tous les fichiers .php du dossier afin de détecter toutes les chaînes encadrées par `_()`.
  Le scan peut comporter des erreurs (en général des accents dans les chaînes à traduire, des accents dans les commentaires du code, des chaînes à traduire vides etc.). Dans ce cas le fichier et la ligne concernée sont indiqués dans le rapport d'erreur. Il est recommandé de corriger l'erreur et recommencer la mise à jour.
  Le rapport affiche les chaînes désuètes (celles qui ne figurent plus dans le code) et les nouvelles chaînes. Les chaînes désuètes sont alors commentées dans le fichier .po et n'apparaissent plus dans l'interface de traduction.

7.1.5.5 Traduction des chaînes

Depuis l'écran du projet dans POEdit, double-cliquer sur le fichier de la langue concernée. La liste des chaînes à traduire apparaît alors. Les nouvelles chaînes sont en premier, les chaînes modifiées en second et les autres chaînes ensuite. Il suffit de cliquer sur une chaîne et entrer la traduction dans le bloc de texte inférieur.

A chaque sauvegarde, le fichier est compilé en un fichier .mo.

L'affichage de l'application affiche alors les chaînes traduites à la place des libellés originaux.

Attention, sur certaines configurations, un redémarrage du serveur web peut être nécessaire pour que les traductions soient mises à jour.

7.1.6 Sphinx

Site officiel du projet Sphinx

7.1.7 Github.com

Pour pouvoir gérer un projet sur ce site, il faut avoir un utilisateur (le bouton "Sign up" de la page d'acucueil permet d'obtenir un compte utilisateur très facilement).

L'objectif d'utiliser ce site est donc de faciliter la contribution à la documentation grâce à l'éditeur en ligne et au système de "Pull Request" et de déclencher simplement la regénération des documentations grâce au lien automatique avec readthedocs.org.

7.1.7.1 Créer un projet

Public(s) concerné(s) : Administrateur de projet openMairie.

Il s'agit ici, de créer un nouveau repository sur github.com dans l'organisation openmairie. Le projet doit s'appeler openlogiciel-documentation (par exemple pour le logiciel openElec "openelec-documentation" et pour le logiciel openRésultat "openresultat-documentation"). Ne rien faire d'autre dans ce repository.

Depuis le tableau de bord de github.com, un clic sur le bouton "+" puis "New repository" en haut à droite de l'écran, à côté du login de l'utilisateur, permet d'accéder au formulaire de création d'un dépôt.

Voici les informations à saisir :
– Owner : sélectionner "openmairie", pour une meilleure lisibilité du projet tous les dépôts doivent être créés dans cette organisation.
– Repository name : le nom doit être logiciel-documentation sans accents sans espaces en minuscules (par exemple pour le logiciel openElec "openelec-documentation" et pour le logiciel openRésultat "openresultat-documentation").
– Description : pour que le dépôt sorte correctement dans les recherches, il faut saisir "Documentation Logiciel Sphinx" (par exemple pour openCimetière "Documentation openCimetière Sphinx").
– Public ou Private : sélectionner "Public" puisque les projets openMairie sont publics.
– Initialize this repository with a README : ne pas cocher la case.
Puis il suffit de cliquer sur le bouton "Create repository".

7.1.7.2 Importer la documentation depuis un projet subversion de l'adullact

Public(s) concerné(s) : Administrateur de projet openMairie.

Les pré-requis sont :
– le projet doit déjà être créé sur github.com
– être dans un terminal pour saisir les commandes suivantes
– les commandes : svn, svn2git et git doivent être installées
Étape 0 : Modifier et définir les variables utilisées dans les commandes suivantes

```
export OLDPRODUCTNAME="openfoncier"
export NEWPRODUCTNAME="openads"
export ADULLACTUSER="fmichon"
```

Étape 1 : Récupération des logs

```
mkdir -p ${NEWPRODUCTNAME}-documentation/${NEWPRODUCTNAME}-documentation && cd ${NEWPRODUCTNAME}-
svn log -q svn://scm.adullact.net/scmrepos/svn/${OLDPRODUCTNAME}/documentation > ../LOG
cat ../LOG | awk -F '|' '/^r/ {sub("^ ", "", $2); sub(" $", "", $2); print $2" = "$2" <"$2">"}' |
cat ../authors-transform.txt
```

Étape 2 : MANUEL - Modification des auteurs

```
=> Il faut faire la correspondance entre les utilisateurs de l'adullact et ceux de github
=> exemple : fmichon = Florent Michon <fmichon@atreal.fr>
=> fmichon = login de l'addulact
=> Florent Michon = nom complet du contributeur
=> fmichon@atreal.fr = mail référencé dans github
vim ../authors-transform.txt
```

Étape 3 : Vérification de l'intervalle

```
export REVS="$(tail -n2 ../LOG|head -n1|awk '{print $1}'|sed "s/r//"):$(head -n2 ../LOG|tail -n1|a
echo $REVS
```

Étape 4 : Récupération de tous les commits (très long)

```
svn2git --authors ../authors-transform.txt --revision $REVS -v svn://scm.adullact.net/scmrepos/svn
```

Étape 5 : Import du code sur github

```
git remote add origin git@github.com:openmairie/${NEWPRODUCTNAME}-documentation.git
git push -u --all
git push --tags
```

Étape 6 : Suppression de l'ancien dépôt de documentation sur l'adullact pour que personne ne committe dessus

```
svn del -m "Déplacement de la documentation vers Github" svn+ssh://${ADULLACTUSER}@scm.adullact.ne
echo "Documentation déplacée vers https://github.com/openmairie/${NEWPRODUCTNAME}-documentation" >
svn import -m "Déplacement de la documentation vers Github" ../MOVED-TO-GITHUB.txt svn+ssh://${ADU
```

7.1.7.3 Faire l'import initial d'un projet sphinx

Public(s) concerné(s) : Administrateur de projet openMairie.

7.1.7.4 Contribuer à une documentation

Public(s) concerné(s) : Contributeur membre du projet openMairie.

7.1.8 readthedocs.org

readthedocs.org est un site qui héberge de la documentation, la rendant accessible et facile à trouver. Il est possible d'importer les documentations sur ce site depuis les système de gestion de version tel que Subversion, Git ou d'autres. Ce site permet de gérer la mise à jour automatique des documentations à chaque commit dans ces systèmes de gestion de version. Le site supporte également le support des versions mais seulement pour Git et non pas pour Subversion à l'heure où cette documentation est rédigée.

L'objectif d'utiliser ce site est donc de ne pas avoir à se soucier de la génération des documentations. C'est ReadTheDcs.org qui s'en occupe et dans tous les formats html, pdf, epub, ...

Pour pouvoir gérer un projet sur ce site, il faut avoir un utilisateur (le bouton "Inscription" en haut à droite de la page d'acucueil permet d'obtenir un compte utilisateur très facilement).

7.1.8.1 Importer un nouveau projet sur RTD

Public(s) concerné(s) : Administrateur de projet openMairie.

Depuis le tableau de bord de readthedocs.org, un clic sur le bouton "Importer", permet d'accéder au formulaire de création d'un projet sphinx existant.

Voici les informations à saisir :
- Nom : le nom du logiciel sans accents sans espaces en minuscules (par exemple : openelec ou openresultat).
- Repo : l'URL de github.com où est stockée le code de la documentation (par exemple : https ://github.com/openmairie/openelec-documentation.git pour openelec ou https ://github.com/openmairie/openresultat-documentation.git pour openresultat).
- Type de dépôt : "Git" puisque le dépôt est sur github.com.
- Description : Le nom du logiciel avec accents avec espaces et avec la casse (par exemple : openElec ou openRésultat).
- Language : "French" puisque la documentation est francophone.
- URL Projet : "http ://www.openmairie.org/".
- Canonical URL : laissons vide pour le moment.
- Single version : ne pas cocher la case.
- Etiquettes : "openmairie".

Puis il suffit de cliquer sur le bouton "Créer".

Si la création du projet s'est bien passée une version de la documentation a du être générée, celle-ci est disponible en cliquant sur le bouton "Afficher les docs" sur la page du projet nouvellement créé.

7.1.8.2 Paramétrer une nouvelle version d'un projet existant

Public(s) concerné(s) : Administrateur de projet openMairie.

Par défaut un projet sur readthedocs.org gère uniquement la dernière version de la cocumentation 'latest' en récupérant la branche par défaut de la documentation sur github.com 'master'.

Il est possible de gérer plusieurs versions de la documentation pour obtenir des URL du style :
– http ://omframework.readthedocs.org/fr/4.2/
– http ://omframework.readthedocs.org/fr/4.4/
– http ://omframework.readthedocs.org/fr/latest/
Chaque version dans readthedocs.org, correspond à une branche dans le dépôt du projet sur github.com.

...

CHAPITRE 8

Contributeurs

(par ordre alphabétique)
– atReal
– Thierry Benita
– Romain Beylerian
– Nicolas Haye
– Nicolas Meucci
– Florent Michon
– Virginie Pihour
– Francois Raynaud
– Sofien Timezouaght